

**ASC X12**

# **X12.71**

---

## **Context Inspired Component Architecture (CICA)**

---

## **Design Rules**

ASC X12

- WORKING DRAFT -

ASC X12  
- WORKING DRAFT -

Copyright © 2003 Data Interchange Standards Association

# TABLE OF CONTENTS

<a href="#"><u>Background</u></a>	8
<a href="#"><u>Part I: Context Inspired Component Architecture</u></a>	9
<a href="#"><u>1 Context Inspired Component Architecture</u></a>	9
<a href="#"><u>1.1 CICA Introduction</u></a>	9
<a href="#"><u>1.2 CICA 101</u></a>	9
<a href="#"><u>2 General Information</u></a>	14
<a href="#"><u>2.1 Business Process</u></a>	14
<a href="#"><u>2.2 Abstract Superclasses, Concrete Subclasses and Usages</u></a>	15
<a href="#"><u>2.2.1 Background</u></a>	15
<a href="#"><u>2.2.2 Abstract Superclasses</u></a>	16
<a href="#"><u>2.3 Invoice Example</u></a>	18
<a href="#"><u>2.4 Structured Rules Overview</u></a>	19
<a href="#"><u>2.4.1 FORM:</u></a>	19
<a href="#"><u>2.4.2 FIT:</u></a>	19
<a href="#"><u>2.4.3 FUNCTION:</u></a>	19
<a href="#"><u>2.5 Detailed Structure Rules</u></a>	20
2.5.1	20
<a href="#"><u>2.5.1.1 Condition 1:</u></a>	20
<a href="#"><u>2.5.1.2 Condition 2:</u></a>	20
<a href="#"><u>2.5.1.3 Condition 3:</u></a>	20
<a href="#"><u>2.5.1.4 Condition 4:</u></a>	20
<a href="#"><u>2.5.1.5 Condition 5:</u></a>	21
<a href="#"><u>2.5.1.6 Condition 6:</u></a>	21
<a href="#"><u>2.5.1.7 Condition 7:</u></a>	21
<a href="#"><u>3 General Rules</u></a>	22
<a href="#"><u>3.1 Terminology</u></a>	22
<a href="#"><u>3.1.1 MUST</u></a>	22
<a href="#"><u>3.1.2 MUST NOT</u></a>	22
<a href="#"><u>3.1.3 SHOULD</u></a>	22
<a href="#"><u>3.1.4 SHOULD NOT</u></a>	22
<a href="#"><u>3.1.5 MAY</u></a>	22
<a href="#"><u>3.2 General Rules</u></a>	22
<a href="#"><u>3.2.1 Requirements Flag</u></a>	22
<a href="#"><u>3.2.2 Minimum Occurs (MinOccurs) values MUST be a non-negative integer equal or greater than zero.</u></a>	23
<a href="#"><u>3.2.3 Maximum Occurs (MaxOccurs) values MUST be either a positive integer greater than zero or a positive infinity represented by "Unbounded."</u></a>	23
<a href="#"><u>3.2.4 Names, XML Names, and Usage Names, and XML Usage Names MUST comply with naming conventions specified in X12.7.</u></a>	23
<a href="#"><u>3.2.5 Content Restriction Flag</u></a>	23
<a href="#"><u>3.2.6 Block Type Flag</u></a>	23
<a href="#"><u>3.2.7 Identification Characteristic Flag</u></a>	24
<a href="#"><u>3.2.8 Abstract Requirements Flag</u></a>	24
<a href="#"><u>3.2.9 Abstract Repeatability Flag</u></a>	24
<a href="#"><u>3.2.10 Slot Detail Flag</u></a>	25
<a href="#"><u>3.2.11 Context Category</u></a>	25
<a href="#"><u>3.2.11.1 The value of the context category MUST be one of the following values:</u></a>	25
<a href="#"><u>3.2.11.2 Context Category Identification</u></a>	25
<a href="#"><u>3.2.11.3 Context Category Identifier Value</u></a>	26

<b>Part II: CICA Constructs</b>	<b>27</b>
<b>4 Core Component Type [CCT]</b>	<b>27</b>
4.1 CCT – Definition	27
4.2 CCT – Content	27
4.3 CCT – Use	29
4.4 CCT – Examples	29
<b>5 Primitives</b>	<b>31</b>
5.1 Primitive – Definition	31
5.2 Primitive – Content	31
5.3 Primitive – Use	31
5.4 Primitive – Examples	31
5.5 Primitive – Design Rules	31
5.5.1 Semantic Rules:	31
5.5.2 Primitive Superclass Property Rules:	32
5.5.2.1 Identifier	32
5.5.2.2 Primitive Superclass Name	32
5.5.2.3 Primitive Description	32
5.5.2.4 Primitive Core Component Type	32
5.5.3 Primitive Subclass Property Rules:	32
5.5.3.1 Primitive Subclass Name	32
5.5.3.2 Content Component Restriction	32
5.5.3.3 Content Component Restriction Value	32
5.5.4 Primitive Usage Rules	32
<b>6 Components</b>	<b>33</b>
6.1 Component – Definition	33
6.2 Component – Content	33
6.3 Component – Use	33
6.4 Component – Examples	33
6.5 Component – Design Rules	34
6.5.1 Component Semantic Rules	34
6.5.2 Component Member Semantic Rules	34
6.5.3 Component Superclass Property Rules	35
6.5.3.1 Component Superclass Name	35
6.5.3.2 Component Description	35
6.5.4 Component Member Properties	35
6.5.4.1 Component Member Usage Name	35
6.5.4.2 Primitive Name	35
6.5.5 Subclass Component Properties	35
6.5.5.1 Component Subclass Name	35
6.5.5.2 Component Subclass XML Name	35
6.5.6 Subclass Component Member Properties	35
6.5.6.1 Primitive Subclass Name	35
6.5.6.2 Requirements Flag	35
6.5.6.3 Minimum Occurs	36
6.5.6.4 Maximum Occurs	36
6.5.6.5 Content Restriction Flag	36
6.5.6.6 Primitive Subclass XML Name	36
<b>7 Blocks</b>	<b>36</b>
7.1 Block - Definition	36
7.2 Block – Content	36
7.3 Block – Use	36
7.4 Block – Examples	37
7.5 Block – Properties	38

7.5.1	<a href="#">Block Semantic Rules</a>	38
7.5.2	<a href="#">Block Member Semantic Rules</a>	38
7.5.3	<a href="#">Block Property Rules, Abstract super class</a>	39
7.5.3.1	<a href="#">BlockName</a>	39
7.5.3.2	<a href="#">BlockXMLName</a>	39
7.5.3.3	<a href="#">BlockType</a>	39
7.5.3.4	<a href="#">BlockDescription</a>	39
7.5.4	<a href="#">Each Block Member Property Rules, Abstract Superclass:</a>	39
7.5.4.1	<a href="#">Position</a>	39
7.5.4.2	<a href="#">Name</a>	39
7.5.4.3	<a href="#">UsageName</a>	39
7.5.4.4	<a href="#">XMLUsageName</a>	39
7.5.4.5	<a href="#">Use</a>	40
7.5.5	<a href="#">Block Subclass Properties</a>	40
7.5.6	<a href="#">Block Member Subclass Properties</a>	40
<b>8</b>	<b><a href="#">Assemblies</a></b>	<b>42</b>
8.1	<a href="#">Assembly - Definition</a>	42
8.2	<a href="#">Assembly – Content</a>	42
8.3	<a href="#">Assembly – Use</a>	42
8.4	<a href="#">Assembly - Examples</a>	42
8.4.1	<a href="#">Assembly Semantic Design Rules</a>	43
8.4.2	<a href="#">An Assembly Member Semantic Rules</a>	43
8.4.3	<a href="#">Assembly Superclass Properties</a>	45
8.4.3.1	<a href="#">AssemblyName</a>	45
8.4.3.2	<a href="#">AssemblyXMLName</a>	45
8.4.3.3	<a href="#">AssemblyDescription</a>	45
8.4.4	<a href="#">Assembly Superclass Member Properties</a>	45
8.4.4.1	<a href="#">Name</a>	45
8.4.4.2	<a href="#">UsageName</a>	45
8.4.4.3	<a href="#">XMLUsageName</a>	45
8.4.4.4	<a href="#">Assembly Type</a>	45
8.4.5	<a href="#">Assembly Subclass Properties</a>	46
8.4.6	<a href="#">Assembly Subclass Member Properties</a>	46
8.4.7	<a href="#">Assembly Design Guidelines</a>	46
<b>9</b>	<b><a href="#">Templates</a></b>	<b>48</b>
9.1	<a href="#">Definition</a>	48
9.2	<a href="#">Content</a>	48
9.3	<a href="#">Use</a>	48
9.4	<a href="#">Examples</a>	49
9.5	<a href="#">Design Rules - Template</a>	49
9.5.1	<a href="#">Template Semantic Design Rules</a>	49
9.5.2	<a href="#">Template Member Semantic Design Rules</a>	50
9.5.3	<a href="#">Semantic Design Rules</a>	50
9.5.4	<a href="#">Superclass Properties Design Rules</a>	51
9.5.4.1	<a href="#">TemplateName</a>	51
9.5.4.2	<a href="#">TemplateXMLName</a>	51
9.5.4.3	<a href="#">TemplateDescription</a>	51
9.5.4.4	<a href="#">TemplateFamily</a>	51
9.5.4.5	<a href="#">BusinessProcess</a>	51
9.5.4.6	<a href="#">BusinessProcessFamily</a>	51
9.5.4.7	<a href="#">BusinessSubprocess</a>	51
9.5.4.8	<a href="#">BusinessSubprocessFamily</a>	52
9.5.4.9	<a href="#">TriggeringEventDescription</a>	52
9.5.4.10	<a href="#">ResponsibleSubCommittee</a>	52

9.5.4.11	<a href="#">Context Category Value List</a>	52
9.5.5	<a href="#">Superclass Member Properties Design Rules</a>	52
9.5.5.1	<a href="#">SlotName</a>	52
9.5.5.2	<a href="#">SlotXMLName</a>	52
9.5.5.3	<a href="#">SlotPurpose</a>	52
9.5.5.4	<a href="#">SlotDetailFlag</a>	52
9.5.5.5	<a href="#">Abstract requirements flag</a>	53
9.5.5.6	<a href="#">Abstract repeatability flag</a>	53
9.5.5.7	<a href="#">Context category value list</a>	53
9.5.5.8	<a href="#">SlotModuleLinkage</a>	53
<b>10</b>	<b><a href="#">Modules</a></b>	<b>55</b>
10.1	<a href="#">Module - Definition</a>	55
10.2	<a href="#">Module – Content</a>	55
10.3	<a href="#">Modules - Use</a>	55
10.4	<a href="#">Modules - Examples</a>	55
10.4.1	<a href="#">Module – Design Rules</a>	56
10.4.2	<a href="#">Semantic Design Rules</a>	56
10.4.2.1	<a href="#">A Module</a>	56
10.4.2.2	<a href="#">A Module Member</a>	56
10.4.3	<a href="#">Syntactic Design Rules</a>	57
10.4.3.1	<a href="#">ModuleName</a>	57
10.4.3.2	<a href="#">ModuleDescription</a>	57
10.4.3.3	<a href="#">Module Type</a>	57
10.4.3.4	<a href="#">ResponsibleSubCommittee</a>	57
10.4.3.5	<a href="#">ContextCategoryValue</a>	57
10.4.3.6	<a href="#">UsageName</a>	57
10.4.3.7	<a href="#">AssemblyorBlock</a>	58
10.4.3.8	<a href="#">RequirementsFlag</a>	58
10.4.3.9	<a href="#">MinOccurs</a>	58
10.4.3.10	<a href="#">MaxOccurs</a>	58
10.4.3.11	<a href="#">Content restriction flag</a>	58
<b>11</b>	<b><a href="#">Documents</a></b>	<b>59</b>
11.1	<a href="#">Definition</a>	59
11.2	<a href="#">Content</a>	59
11.3	<a href="#">Use</a>	59
11.4	<a href="#">Examples</a>	59
11.5	<a href="#">Design Rules – Document</a>	59
11.5.1	<a href="#">Semantic Design Rules</a>	59
11.5.2	<a href="#">Syntactic Design Rules</a>	60
11.5.2.1	<a href="#">DocumentName</a>	60
11.5.2.2	<a href="#">DocumentXMLName</a>	60
11.5.2.3	<a href="#">DocumentDescription</a>	60
11.5.2.4	<a href="#">DetailMaxOccurs</a>	60
11.5.2.5	<a href="#">ResponsibleSubCommittee</a>	60
11.5.2.6	<a href="#">ContextCategoryValue</a>	61
11.5.2.7	<a href="#">DocumentSlot</a>	61
11.5.2.8	<a href="#">ModuleSlotXMLName</a>	61
11.5.2.9	<a href="#">ModuleName</a>	61
11.5.2.10	<a href="#">ModuleUsageName</a>	61
11.5.2.11	<a href="#">ContextCategory</a>	62
11.5.2.12	<a href="#">ContextCategoryValue</a>	62
11.5.2.13	<a href="#">ReqDesignator</a>	62
11.5.2.14	<a href="#">MinOccurs</a>	62
11.5.2.15	<a href="#">MaxOccurs</a>	62

---

<a href="#"><u>APPENDIX A: CORE COMPONENTS CONTEXT CATEGORIES</u></a>	64
-----------------------------------------------------------------------	----

ASC X12  
- WORKING DRAFT -

## DESIGN RULES

### Background

These design rules have been developed to assist in establishing uniformity in X12 XML business document development and maintenance efforts. The design rules are to be used as stated. There shall be no deviation from them, with all new development and maintenance expected to utilize them.

These design rules uses as its basis the philosophical foundation and general design principles forwarded in the published technical report, ASC X12 Reference Model: Context Inspired Component Architecture (CICA).

This document is intended to be flexible as new situations arise that require new rules or guidelines, or modifications to existing ones. Therefore, individuals involved with XML business document development and maintenance are urged to keep the latest edition of the Design Rules document handy for reference. Modifications and suggestions for improvement of this guideline are welcome and should be sent to Task Group 1 of Subcommittee J, Technical Assessment, care of:

**Secretariat, ASC X12**

Data Interchange Standards Association, Inc.  
7600 Leesburg Pike • Suite 430  
Falls Church, VA 22043-2004  
(703) 970-4480

ASC X12  
- WORKING DRAFT -



# Part I: Context Inspired Component Architecture

## 1 Context Inspired Component Architecture

### 1.1 CICA Introduction

The Context Inspired Component Architecture (CICA) offers a method for building electronic business messages using XML. CICA takes the lessons from two decades of EDI experience, and updates that experience to take advantage of XML's flexibility.

Like EDI, CICA presents a standard structure for business messages, and adds in the implementation details required by different industries. However, CICA also allows organizations to apply the same methods for messages used across industries. Thus organizations can relate the terms used in their own industries to potential trading partners in other lines of business. To achieve these somewhat contradictory objectives (consistent structure, with flexibility) presents technical challenges. CICA needs to manage a much larger quantity of information in various contexts, while maintaining a consistent approach to encourage interoperability across those contexts.

The purpose of this section is to provide an introduction to the CICA. It will explain the most basic and important concepts behind the framework and its components. In an attempt to convey the core concepts, we may at times overly simplify some of the explanations. However, with a strong grounding in the core concepts, a more detailed and complex level of understanding can ultimately be achieved.

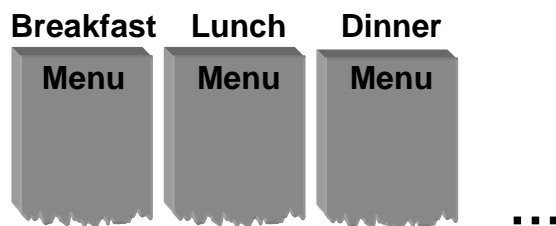
The general framework of the CICA can be simply stated. A template is created for a business process. A template consists of slots. The slots may be filled with a number of different modules. Modules may be broken down into a number of sections including assemblies, blocks, components and primitives. A unique instance of the completed framework is called a document.

### 1.2 CICA 101

In order to illustrate some of the concepts and relationships, we will use an analogy that many people are familiar with, a restaurant menu.

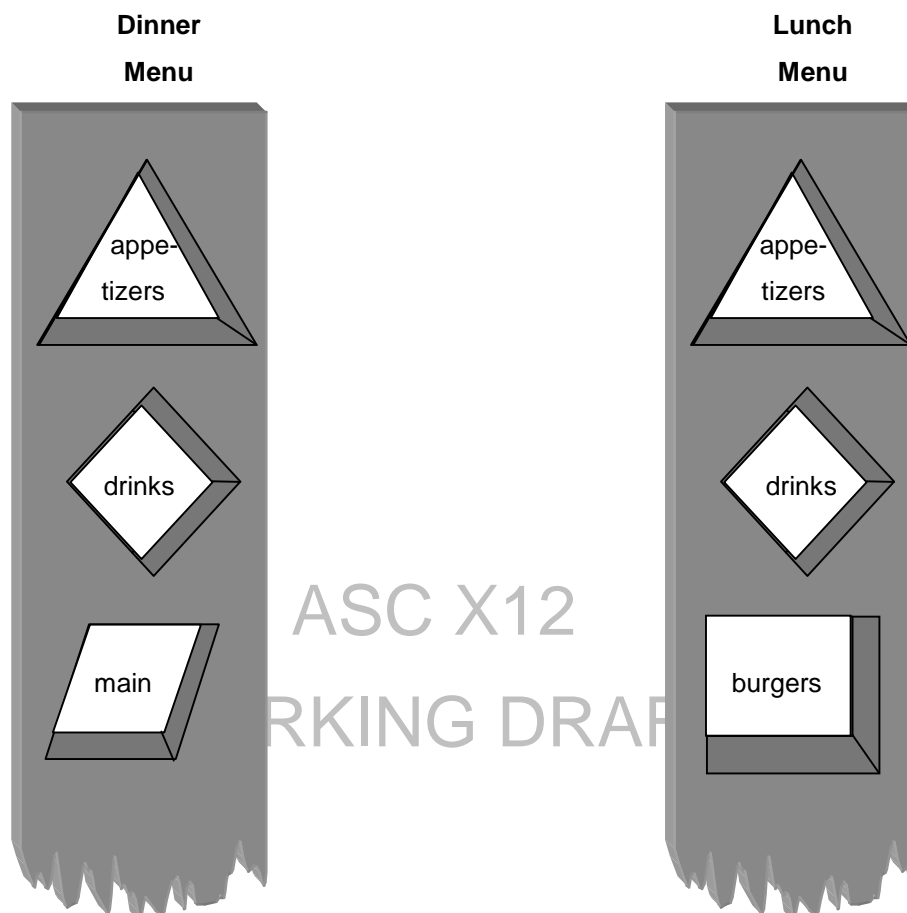
Many industries face the need to organize and present information in a structure, while allowing for maximum flexibility. In a restaurant, this need expresses itself in creating a menu for its customers. Almost every restaurant, from the corner diner to the gourmet establishment, offers a variety of items. The issue facing the restaurant is how to organize this list of items to be meaningful to the customers. The restaurant could organize food items in a number of different ways: alphabetically, by the main ingredients, by the type of dish offered, by the meal in which they are offered, or the course within the meal.

The restaurant might begin by breaking down its offerings into three distinct menus, one each for breakfast, lunch, and dinner. In the CICA context, these menus are analogous to templates. They are similar in that they convey the food offerings to the customer, but are different as they reflect the food preferences generally associated with the different times of day (Figure 1).



*Figure 1. Basic Menu Structure*

Within each menu, one can identify different courses. The courses might include an appetizer, main dish, dessert, and beverage. In the CICA context, slots are analogous to the courses reflected on the menu: a slot for appetizer, main dish, dessert, beverage, etc. Some of these may be shared across the different menus. For example, the both the dinner template and the breakfast would probably have a slot for main dish, where as the breakfast template would probably not have a dessert slot. Both may have a beverage slot (Figure 2).



*Figure 2. Menu Organization*

Types of dishes or course selection within a meal can include meat, fish, vegetarian main dishes or hot or cold appetizers. Under each of these categories then come the individual selections. The individual dishes that we use to fill the main dish slot are similar to modules. Each course slot may be filled with a number of modules that are appropriate to the menu template.

But the breakdowns do not end at the individual selections. The selections will generally have prices associated with them as well as more detailed information about the selection, for example a restaurant may offer appetizers, soup, vegetables, and a salad with the main course. In other establishments, patrons can specify the type of preparation or amount of seasoning they prefer, or they can indicate substitutes if on a restricted diet. Even on a side dish, the choices can proliferate, for instance selecting the type of dressing on a salad (Figure 3).

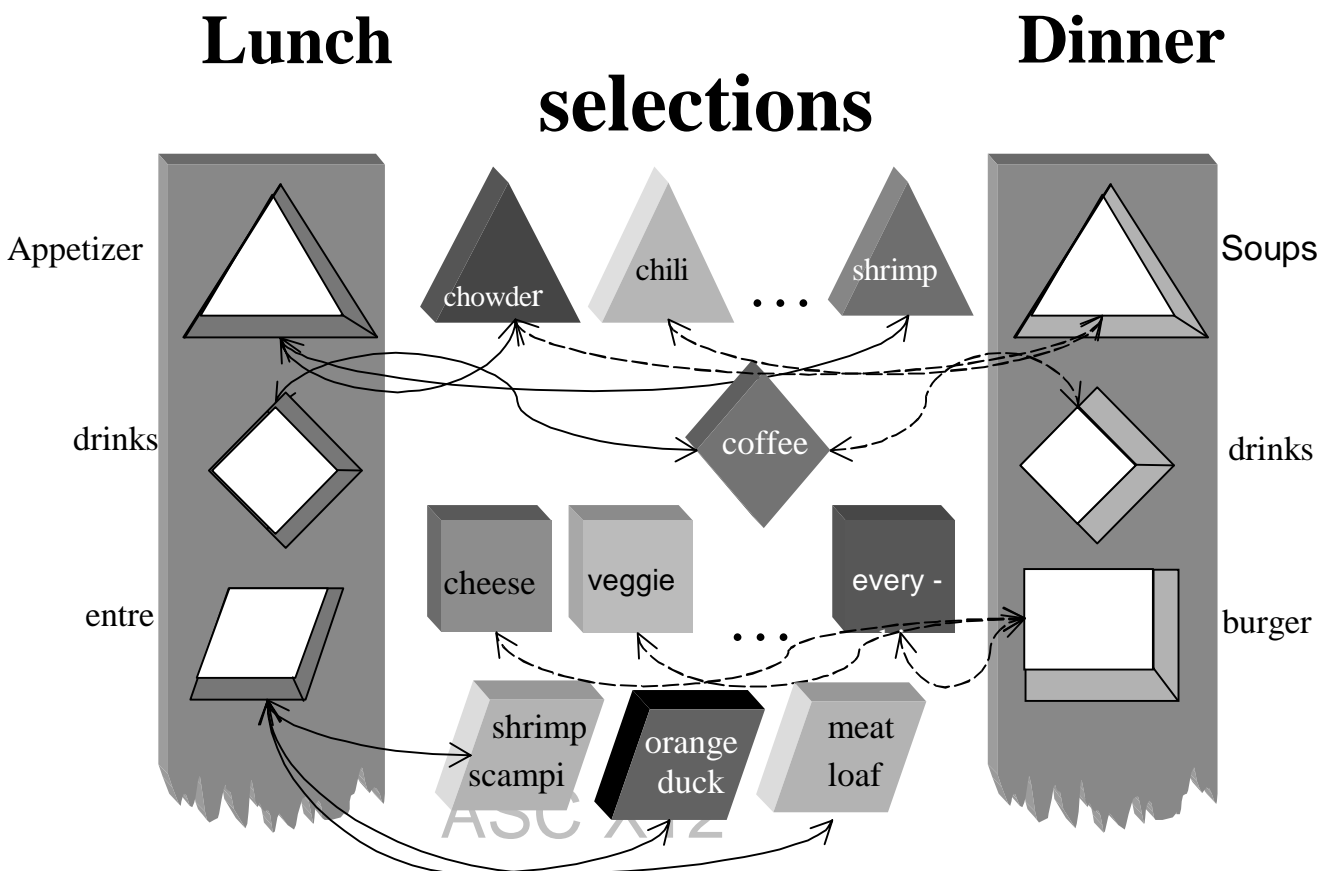
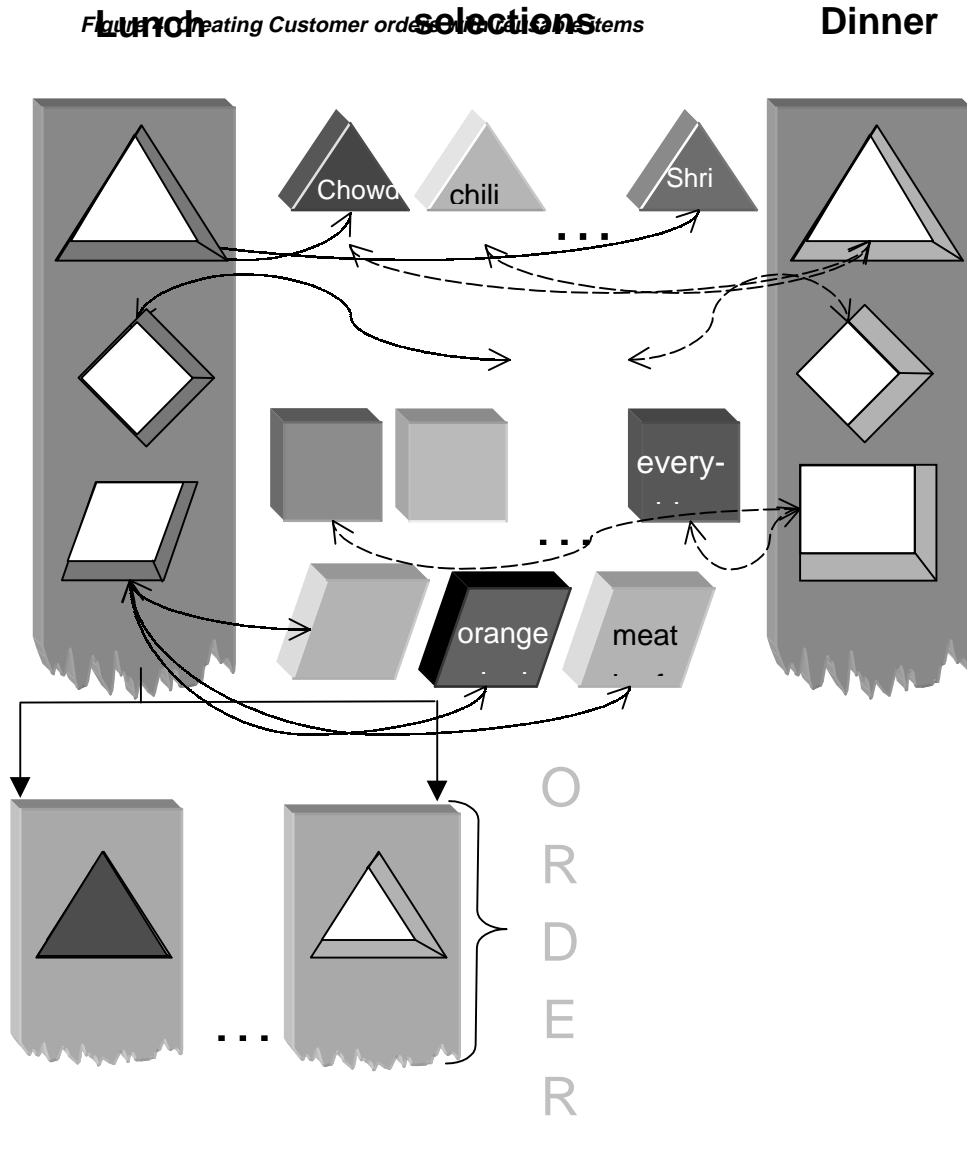


Figure 3. Reusing menu selections

The server, in taking the order will list the selections chosen by the customer, and create an individual order. In many cases, the different choices can be mixed from one order to another, thus orders will have different appetizers, soups, salads, main course, and beverages, or just soup and salad, and a main course, or just soup and salad. The idea is to build a meal with interchangeable menu items or components (Figure 4, next page).



While writing a restaurant or food service menu may seem light years away from building electronic business messages, many of the same principles apply:

- A disciplined underlying structure – menus for all of the meals follow the same basic hierarchy, making it easier to generate new menus
- Encouragement of item reuse – e.g., the side-salad with dinner is also offered at lunch; the cooks can use the same ingredients and methods, thus simplifying their work
- A structured approach to reuse – cooks planning the meals know in advance the extent of reuse available, and servers know the limits of substitutions; for example, breakfast and lunch generally have fewer courses than dinner, limiting the items available at the earlier meals

With electronic business messages, these same principles can achieve comparable benefits in organization and simplification. A disciplined underlying structure provides a basic hierarchy for electronic business messages, much like it does for menu planners. The

structure provides the cook with a way of organizing menus, but does not limit the creativity of the final product. Likewise with e-business messages, having an underlying structure offers a way for trading partners to consistently organize the data they exchange, but does not limit the kinds of business that the trading partners can undertake.

The reuse of items enables both the menu planners and e-business message architect to simplify their efforts and make better use of their resources. If the food service planners can provide to customers a wide selection of items by offering variations on a basic set of ingredients, they meet the customers' needs for variety while simplifying their work and probably making better use of their time. With e-business messages, the use of interchangeable data items offers a way of making maximum use of IT resources and simplifying the message contents. The CICA architecture builds electronic business messages by choosing from the interchangeable or reusable items and combining them at different levels.

In building business messages, the message uses a process analogous to the restaurant menu. The message collects components, important pieces of information, needed by the trading partner to perform actions beneficial to the parties to the transaction. In most cases, the industry in which the companies operate will have processes and terminology defined, much like the way the business environment of a food service defines the meals it serves: a fancy expense-account restaurant will not likely serve breakfast, for example.

One can draw parallels between the hierarchy in restaurant menus and the CICA architecture:

Menu hierarchy	CICA architecture
Customer order (baked salmon, ranch dressing on salad)	Document (event invoice goods)
Menu (lunch menu)	Template (event invoice)
Menu category (lunch, main course)	Template slot (Line Product)
Menu selection (baked salmon, includes side salad)	Module (Catalog Goods)
Dish combination (baked salmon plus side salad)	Assembly (Commodity Goods)
Completed dish (baked salmon)	Block (Product)
Food item (salmon)	Component (Product ID)
Item ingredient (dressing on salad)	Primitive (ID Number Type)

In both cases, the structure defines an overall hierarchy for the operators to organize their menus on one hand or goods invoice on the other. The interchangeable parts make it possible to reuse items on different menus or in other business messages. A company could use the unit price block, for example, in various messages, such as quotation or purchase order, as well as invoice.

## 2 General Information

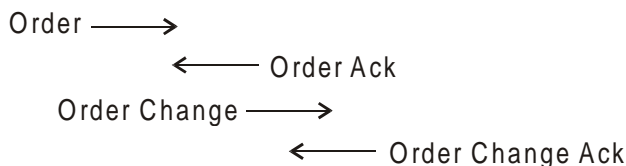
The information in this section applies to the entire document, and is placed in this section to assist in the general overview. The information may repeat in subsequent sections.

### 2.1 Business Process

In CICA, the use of 'business process' is used in the most general sense of the term. In this context, business processes are general from two different perspectives. Business processes are both industry independent and independent of specific message flows. Business processes are bigger than a message, that is, completion of a sequence of steps is required to constitute a business exchange. This does not, however, require that a business process include multiple business documents, it simply acknowledges the fact that some business exchanges use a combination of external mechanisms with 'silence equaling acceptance' an understood part of the business process.

In this context, business processes refer to 'door-to-door' processes limited to the public or exposed portions of the process external to the organization's firewall, if you will. When an organization receives a request, it might carry out any number of internal steps to determine, for instance, whether it wants to do business with a trading partner, or whether it can do business with that trading partner. The steps an organization takes to make such decisions are internal and proprietary to the organization often resulting in competitive advantage. These internal steps are not considered within the scope of the CICA definition for business process. Of course, once a decision has been made, the organization's response as to the request is very much a part of the business process.

This leads to the balance between how general and how specific. In other words, when are two business processes 'the same' and when are they 'different'. The question is illustrated in the following set of messages.



From this small set of messages, several specific sequences can be produced, as follows.

1. Order, Silent Ack, done
2. Order, Order Ack, done
3. Order, Order Change, done
4. Order, Order Ack, Order Change, Order Change Ack

As illustrated above, even when there are a variety of sequences that can be followed it does not necessarily represent a different order process. Conversely, having the same sequence of messages does not imply the same business process, for example a situation with orders between a production supplier and its customers versus casual orders between trading partners without a prior business relationship. So, where does that leave us? A set sequence of messages does not guarantee us a unique business process action. A different sequence of messages does not guarantee us that a message is different. So, what does? A set of indicators,

- Pre conditions/state
- Trigger Event

- Post conditions/state

together act as indicators determining whether a message is unique, or the same as other business process usages. This test helps determine the set of UNIQUE messages, based on business process. For each of these, in CICA, Templates are created that is, they are specific to a set of key indicators that are used to determine use/uniqueness.

This is a significant departure from X12 current EDI practices. The X12 philosophy is to have a single transaction set for each business purpose or action for example a single Invoice. While in practice this is not actually done, it remains the prevailing X12 philosophy. CICA recognizes the value of and has as its objective to utilize a single means to fulfill a single purpose. Dissimilarities create complexity, which CICA is equally committed to minimize.

Two primary factors lead to complexity in X12's EDI Transaction Sets:

1. Supporting the needs for multiple business processes within the same Business Document.

General business functions, such as invoicing, are perceived by many to be the same business process – an invoice is an invoice. This is not necessarily the case with the differences in business circumstances associated with the generation of one invoice can be different than that of another, affecting the high level organization. Having to support dual business processes with the use of a single business document leads to ambiguous business message design often resulting in more than one position to place the same information. This resulting complexity is dealt with by CICA in the following ways:

- a. Each business process different need for a different high level organization gets a different Template
  - b. Templates only contain Slots, and loose Modules are constructed to fit into those Slots. This allows the same Module to be used in multiple Templates, minimizing the cost of having Templates. This design is dramatically different for current EDI practices.
2. Differences in information needs due to the differences in products and/or services that are the subject of the business document.

CICA has an elegant solution for this, allowing support for details sufficient to meet the needs of implementation.

- a. Modules can be built out of reusable constructs, which can be fully customized to meet the needs of each user.
- b. Modules coexist within CICA in that multiple Modules are linked to the same Slot, allowing for peer Modules to be associated with the same role or purpose.
- c. This granularity approach further provide for the natural information flow through a business process, where the same information appears within multiple messages within the process. The same Module therefore can be used in many business documents.

## 2.2 Abstract Superclasses, Concrete Subclasses and Usages

### 2.2.1 Background

Important to the X12 community is to maximize the efficiency of reuse. This was a critical design goal in CICA. One of the mechanisms in CICA to support this design goal is the use of layering to make optimum use of reuse. One such case is the use of Abstract Superclasses and Concrete Subclasses.

The concept is simple. CICA include the set of details necessary for implementation, such as Minimum Length, Maximum Length, Requirement Designation, etc. For the same basic structure, these details can vary significantly, determined by implementation needs. Abstract Superclasses with Concrete Subclasses allow CICA to manage both levels of requirements.

## 2.2.2 Abstract Superclasses

Abstract Superclasses embody the structure of the construct. For example, a Person Party Block Abstract Superclass might be as shown below.

### Abstract Super Class

ASClass Person Party Block			
PersonName Component			
Usage		Primitive	
FirstName		NameText	
MiddleName		NameText	
LastName		NameText	

For that Abstract Superclass, one or more Concrete Subclasses will be specified. There might be a Concrete Subclass with a set of restrictive properties specified, as shown below:

### Concrete Subclass

ASClass Person Party Block, CSCClass Person Party Block 01					
PersonName Component					
Usage	Primitive	Min	Max	Req	
FirstName	NameText	1	20	O	
MiddleName	NameText	1	1	O	
LastName	NameText	1	60	M	

Now, with the full construction specified [all of the elements, plus the restrictive details], the Block is available to be used in a Module.

When the Block is used, a usage name is specified. One or more usages can reuse the same concrete subclass. Examples could be as follows:



**Usage**

Usage: Patient						
	<b>ASClass Person Party Block</b>					
		<b>PersonName Component</b>				
		<b>Usage</b>	<b>Primitive</b>	<b>Min</b>	<b>Max</b>	<b>Req</b>
		FirstName	NameText	1	20	O
		MiddleName	NameText	1	1	O
		LastName	NameText	1	60	M

Usage: Criminal						
	<b>ASClass Person Party Block</b>					
		<b>PersonName Component</b>				
		<b>Usage</b>	<b>Primitive</b>	<b>Min</b>	<b>Max</b>	<b>Req</b>
		FirstName	NameText	1	20	O
		MiddleName	NameText	1	1	O
		LastName	NameText	1	60	M

ASC X12

- WORKING DRAFT -

## 2.3 Invoice Example

For the purposes of this document, a complete, top to bottom example has been constructed in order to illustrate the concepts. Note that each box is numbered in order to easily refer to subsection within the example.

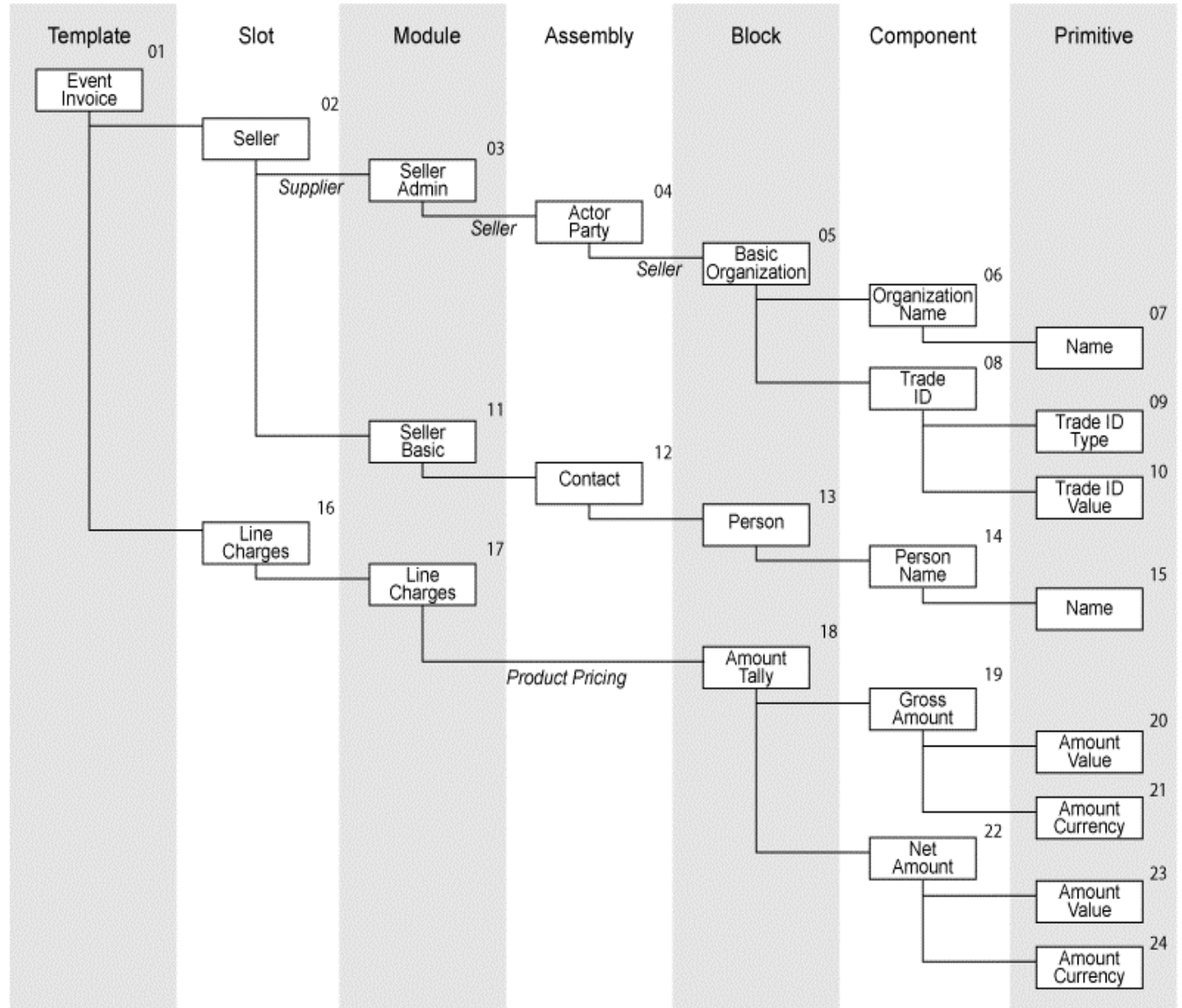


Figure 5. Example

## 2.4 Structured Rules Overview

There are three tests that can be applied when comparing two candidate information constructs to determine the level to which they are related. These are Form, Fit and Function, and they are taken from the Parts world where they are used to determine when a new part number needs to be assigned. These tests, while the same for each CICA construct, have slightly different implications depending on the semantic abstraction of the construct. Modules, the most semantically specific construct, are more sensitive to purpose and usage and a little less impacted by structure. In contrast, Blocks contain abstract semantics and are affected more by structure. These details affect how to apply these tests and the resulting rules. The general concepts are presented below.

For eBusiness considerations, Form, Fit and Function are defined as follows:

### 2.4.1 FORM:

Physical – the structure, contents and components of the information structures being specified. For example, parts have names and so do people. People have first, middle and last names, whereas a part has a single name, part name. The difference in Form makes these two types of names different. In contrast, you might have a Student First Name and Student Last Name, compared with a Patient First Name and Patient Last Name. Form-wise, these two examples are the same.

### 2.4.2 FIT:

Identity-Meaning-Specificity – Two organizations or industries that share the common element named Part Number have reason to believe that there is some commonality. Sometimes two uses of an identically named item do not provide the same level of specificity, and therefore these items are not the same thing. In ebXML, a case using a Vehicle Identification Number, "VIN" was used. Different organizations use the VIN, but they may be referring to a different subset of sub-components. Each subsection of the component parts of the VIN, for the same vehicle, is different information. Can all of these different subsets of the same base number all be called VIN – no! Other examples are found with Part Number, with different levels of specificity found with a construct called Part Number. For these to be considered the same, they must specify the same level of specificity.

### 2.4.3 FUNCTION:

Purpose or how used. – When comparing two information constructs, such as Product, there is a common purpose or usage – which motivates treating them as 'the same', even though the detail used to specify various Products can vary widely. In some cases, the various Product descriptions are similar in form, but in many instances, this is not the case. Efforts to merge dissimilar definitions results in ambiguity, which later needs to be disambiguated. In the CICA architecture, through the use of abstract layers and links, these Functionally-related constructs are associated, without imposing ambiguous merging. The product specification for a Widget is dramatically different from the product specification for Visiting Healthcare services.

## 2.5 Detailed Structure Rules

### 2.5.1

The levels of equality that are true determines how closely related two information constructs are. Consider the following:

#### 2.5.1.1 Condition 1: FORM = YES

FIT = YES

FUNCTION = YES

When all three tests are true, then with 100% certainty we can determine that the two are the same thing, that is, the constructs are semantically equal. Examples of this situation are Shipper, Seller, or Supplier. These are different industry-specific terms for a semantically equivalent party playing a role. Frequently the descriptive details are exactly the same; and when that case is true, they are semantic equals in every sense.

#### 2.5.1.2 Condition 2: FORM = NO

FIT = NO

FUNCTION = YES

When equality is based on function alone, the two information constructs appear below a common parent structure. For example, in the travel industry you have rooms in hotels and passenger seats on flights. Although they are specified with different data elements and are called different things, they are used in the same manner in a business process/message. Thus, the two appear beneath a common parent [at some level], possibly human service products.

#### 2.5.1.3 Condition 3: FORM = YES

FIT = NO

FUNCTION = NO

This case is very common in EDI today, and is well supported. The X12 N1 loop specifies the name, ID and address of any party, person or organization. The fundamental difference is that in the CICA architecture, Blocks are specified for the various data arrangements [different where a party is an individual versus an organization]. Further, this is independent of whether the construct can represent many purposes, which is the expected case. Therefore, in terms of Blocks, it is expected to have a single block [Party with First, Middle and Last Name] used for many specific parties: Passenger, Patient, or Student.

#### 2.5.1.4 Condition 4: FORM = NO

FIT = YES

FUNCTION = NO

This is the case where an information construct serves the same semantics in two different settings/business conditions, but it is used differently and has different components.

**2.5.1.5 Condition 5:**

FORM = YES

FIT = NO

FUNCTION = YES

In the automotive industry, Part Number is used to specify the desired product.

Manufacturer A has a significant digit part number which is really a composite of several identifiers: base + change number + color number + location on vehicle + etc.

Manufacturer B and others have a part number too, but it refers only to the base. Separate additional values are required which include: change number, color number, location on vehicle, etc.

Both of these are related in that they are used to specify THE part, but they are NOT semantically equal. They do not provide the same level of specificity. Therefore, although they are used for the same base purpose, they cannot be used interchangeably and therefore, they are not the same.

**2.5.1.6 Condition 6:**

FORM = YES

FIT = YES

FUNCTION = NO

This case happens primarily when multiple business processes are involved. Consider a scenario where a Doctor is treating Patients versus a scenario of a business process where a Clinic is communicating its Assets – its staff. In both cases the form and fit are the same, but the function is different. It is unclear what structural implications this case has.

**2.5.1.7 Condition 7:**

FORM = NO

FIT = YES

FUNCTION = YES

In this case there, is a difference in form, as is the case with Person Name versus Organization Name. Both cases are serving the function to specify the Party. Last Name does not equal Organization Name, because they don't deliver the same level of precision. In order to achieve the same level of "Fit", it is Organization Name = Last Name + Middle Name + First Name. Fit ensures semantic equality.

## 3 General Rules

### 3.1 Terminology

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL**, when they appear in this document, are to be interpreted as described in [RFC2119] as quoted here:

**NOTE:**

The emphasis of these words is influenced by the requirement level of the document in which they are used.

#### 3.1.1 **MUST**

This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

#### 3.1.2 **MUST NOT**

This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

#### 3.1.3 **SHOULD**

This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

#### 3.1.4 **SHOULD NOT**

This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

#### 3.1.5 **MAY**

This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, **MUST** be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, **MUST** be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

### 3.2 General Rules

#### 3.2.1 **Requirements Flag**

The requirements flag indicates that the presence of the member within its parent entity is one of mandatory, optional, or not used.

The requirements flag **MUST** have one of the following values:

**M** = Mandatory, the member **MUST** be present

**O** = Optional, the member **MAY** be present

**N** = Not Used, the member is not used in a particular instance

The value of the requirements flag **MUST** be consistent with the min occurs.

If the value of the requirements flag is mandatory (M) then the value of the min occurs **MUST** be one or more.

If the value of the requirements flag is optional (O) then the value of the min occurs may be 0 or more.

The requirements flag **MUST** be used on the members of the concrete subclasses component-subclass, block-subclass, assembly-subclass, and module.

### 3.2.2 **Minimum Occurs (MinOccurs) values **MUST** be a non-negative integer equal or greater than zero.**

The min occurs **MUST** be used on the members of the concrete subclasses component-subclass, block-subclass, assembly-subclass, and module.

### 3.2.3 **Maximum Occurs (MaxOccurs) values **MUST** be either a positive integer greater than zero or a positive infinity represented by “Unbounded.”**

The max occurs **MUST** be used on the members of the concrete subclasses component-subclass, block-subclass, assembly-subclass, and module.

### 3.2.4 **Names, XML Names, and Usage Names, and XML Usage Names **MUST** comply with naming conventions specified in X12.7.**

### 3.2.5 **Content Restriction Flag**

[Editor's Note: An example would be helpful here]

The content restriction flag identifies that the member is part of a unique subset of the parent entity where the subset members share a common content restriction. A parent entity **MUST** have no more than one content-restricted subset from the set of its members.

If there is no content restriction, the content restriction flag **MAY** have no value,

If there is content restriction, the content restriction flag **MUST** have one of the following values:

**X** = Exclusive, one and only one member of the subset **MUST** be present

**A** = Inclusive, at least one of one member of the subset **MAY** be present

The value of the content restriction flag **MUST** be consistent with the value of the requirements flag. If there is a content restriction, the requirements flag **MUST** indicate the member is optional.

### 3.2.6 **Block Type Flag**

The block type flag Identifies the type of block.

The block type flag **MUST** have one of the following values:

**P** = Party

**L** = Location

**E** = Event

**R** = Resource

The block flag **MUST** be used only on the abstract superclass block. It is not used with any other structure.

### 3.2.7 Identification Characteristic Flag

The identification characteristic flag identifies the nature of a block member.

The values for the identification characteristic flag **MUST** be one of the following:

**I** = Identification

**C** = Characteristic

The identification characteristic flag **MUST** be used only on the members of the abstract superclass block.

### 3.2.8 Abstract Requirements Flag

The abstract requirements flag indicates that the presence of the member within its parent entity is either mandatory or optional.

The abstract requirements flag **MUST** have one of the following values:

**M** = Mandatory, the member **MUST** be present

**O** = Optional, the member **MAY** be present

The value of the abstract requirements flag **MUST** be consistent with the min occurs.

If the value of the abstract requirements flag is mandatory (M) then the value of the min occurs **MUST** be greater than one.

If the value of the min occurs is equal to zero, then the value of the abstract requirements flag **MUST** be optional (O).

The abstract requirement flag **MUST** be used only on the members of the abstract superclass template.

### 3.2.9 Abstract Repeatability Flag.

The abstract repeatability flag indicates that the member may be present one or multiple times.

The value of the abstract repeatability flag **MUST** be one of the following values:

**S** = Singular, the member **MUST** appear once and only once.

**M** = Multiple, the member **MAY** appear one or more times

The value of the abstract repeatability flag **MAY** be affected by the requirements flag also associated with the member.

The abstract repeatability flag **MUST** be used only on the members of the abstract superclass template.



### 3.2.10 Slot Detail Flag

The slot detail flag indicates if the slot is part of the detail section of the template. All modules that fill slots with the slot detail flag set to detail will repeat as a group

If the member is not part of the detail section the value of the slot detail flag MAY have no value.

The value of the slot detail flag MUST be one of the following values:

**D** = Detail, the member is part of the detail section

The slot detail flag MUST only be used on the members of the abstract superclass template.

### 3.2.11 Context Category

The context category indicates the business context as defined in the CCTS as defined in section 6.2.2 of the ebXML Core Components Technical Specification.

As defined in the CCTS, there are three (3) pieces of information necessary to specify the Context, the Context Category, the Context Category Identification Scheme, and the Context Identifier Value.

#### 3.2.11.1 The value of the context category MUST be one of the following values:

**BP** = Business Process

**PC** = Product Classification

**IC** = Industry Classification

**GP** = Geopolitical

**OC** = Official Constraints

**BR** = Business Process Role

**SR** = Supporting Role

**SC** = System Capabilities

#### 3.2.11.2 Context Category Identification

The name of the identification scheme as specified in section 6.6.2 of the CTS {forward reference}.

This space contains the names of the identification schemes referenced in the CCTS {reference}. However, some of the context categories do not identify identification schemes, so those Context Categories with specified identification schemes are listed below, those without defined identification schemes are indicated with the wild card asterisk.

"UN/CEFACT Catalogue of Common Business Processes"

"Universal Standard Product and Service Specification (UNSPSC)"

"Standard International Trade Classification (SITC Rev .3)"

"Harmonized Commodity Description and Coding System (HS)"

"Classification Of the purposes of non Profit Institutions serving households (COP1)"

"International Standard Industrial Classification (ISIC)"

"Universal Standard Product and Service Specification (UNSPSC) Global"

"ISO 3166"

"UN/EDIFACT Code List for DE 3035 Party Roles"

\*

**3.2.11.3 Context Category Identifier Value**

The value within the identification scheme that specifies the aspect of Context.

ASC X12  
- WORKING DRAFT -

## Part II: CICA Constructs

### 4 Core Component Type [CCT]

#### 4.1 CCT – Definition

Core Component Type (CCT) is defined within the Core Components Technical Specification (CCTS). The CCT is the base of CICA and used in its entirety within the architecture. A CCT consists a single content component that carries the actual content plus one or more supplementary components that give extra definition to the content component. Core Component Types do not have business semantics.

For example, the CCT of Amount.type, the content component carries the value of 12. This value has no meaning by itself. But if the supplementary component, dollars, is added the CCT does have real semantic meaning.

The core component types are finite in quantity and are specifically defined in this Design Rules document.

#### 4.2 CCT – Content

CICA utilizes the CCTS listing of CCTs. Each CCT contains a single value CCTC, where the actual business information is stored. In addition, one or more supplementary CCTCs are defined for each CCT. The table is as follows:

CCT Dictionary Entry Name	Definition	Remarks	CCTC Components
<b>Amount. Type</b>	A number of monetary units specified in a currency where the unit of currency is explicit or implied.		<ul style="list-style-type: none"> <li>Amount. Content</li> <li>Amount Currency. Identifier</li> <li>Amount Currency. Code List Version. Identifier</li> </ul>
<b>Binary Object. Type</b>	A set of finite-length sequences of binary octets.	Shall also be used for <i>Data Types</i> representing graphics (i.e., diagram, graph, mathematical curves or similar representations), pictures (i.e. visual representation of a person, object, or scene), sound, video, etc.	<ul style="list-style-type: none"> <li>Binary Object. Content</li> <li>Binary Object. Format. Text</li> <li>Binary Object. Mime. Code</li> <li>Binary Object. Encoding. Code</li> <li>Binary Object. Character Set. Code</li> <li>Binary Object. Uniform Resource. Identifier</li> <li>Binary Object. Filename. Text</li> </ul>

CCT Dictionary Entry Name	Definition	Remarks	CCT Components
<b>Code. Type</b>	A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an <i>Attribute</i> together with relevant supplementary information.	Should not be used if the character string identifies an instance of an <i>Object Class</i> or an object in the real world, in which case the Identifier. Type should be used.	<ul style="list-style-type: none"> <li>• Code. Content</li> <li>• Code List. Identifier</li> <li>• Code List. Agency. Identifier</li> <li>• Code List. Agency Name. Text</li> <li>• Code List. Name. Text</li> <li>• Code List. Version. Identifier</li> <li>• Code. Name. Text</li> <li>• Language. Identifier</li> <li>• Code List. Uniform Resource. Identifier</li> <li>• Code List Scheme. Uniform Resource. Identifier</li> </ul>
<b>Date Time. Type</b>	A particular point in the progression of time together with relevant supplementary information.	Can be used for a date and/or time.	<ul style="list-style-type: none"> <li>• Date Time. Content</li> <li>• Date Time. Format. Text</li> </ul>
<b>Identifier. Type</b>	A character string to identify and distinguish uniquely, one instance of an object in an identification scheme from all other objects in the same scheme together with relevant supplementary information.		<ul style="list-style-type: none"> <li>• Identifier. Content</li> <li>• Identification Scheme. Identifier</li> <li>• Identification Scheme. Name. Text</li> <li>• Identification Scheme Agency. Identifier</li> <li>• Identification Scheme. Agency Name. Text</li> <li>• Identification Scheme. Version. Identifier</li> <li>• Identification Scheme Data. Uniform Resource. Identifier</li> <li>• Identification Scheme. Uniform Resource. Identifier</li> </ul>
<b>Indicator. Type</b>	A list of two mutually exclusive Boolean values that express the only possible states of a <i>Property</i> .		<ul style="list-style-type: none"> <li>• Indicator. Content</li> <li>• Indicator. Format. Text</li> </ul>

<b>Measure. Type</b>	A numeric value determined by measuring an object along with the specified unit of measure.		<ul style="list-style-type: none"> <li>• Measure. Content</li> <li>• Measure Unit. Code</li> <li>• Measure Unit. Code List Version. Identifier</li> </ul>
----------------------	---------------------------------------------------------------------------------------------	--	-----------------------------------------------------------------------------------------------------------------------------------------------------------

CCT Dictionary Entry Name	Definition	Remarks	CCT Components
<b>Numeric. Type</b>	Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity or unit of measure.	May or may not be decimal	<ul style="list-style-type: none"> <li>• Numeric. Content</li> <li>• Numeric. Format. Text</li> </ul>
<b>Quantity. Type</b>	A counted number of non-monetary units possibly including fractions.		<ul style="list-style-type: none"> <li>• Quantity. Content</li> <li>• Quantity. Unit. Code</li> <li>• Quantity Unit. Code List. Identifier</li> <li>• Quantity Unit. Code List Agency. Identifier</li> <li>• Quantity Unit. Code List Agency Name. Text</li> </ul>
<b>Text. Type</b>	A character string (i.e. a finite set of characters) generally in the form of words of a language.	Shall also be used for names (i.e. word or phrase that constitutes the distinctive designation of a person, place, thing or concept).	<ul style="list-style-type: none"> <li>• Text. Content</li> <li>• Language. Identifier</li> <li>• Language. Locale. Identifier</li> </ul>

### 4.3 CCT – Use

The CCT is used as the basis for deriving a set of CICA Primitives. The CCT is very general, almost a data types, yet at a more semantic level. Therefore, the CCT acts as semantic anchor point within the CICA architecture linking all usages and derivations

### 4.4 CCT – Examples

CCT for Amount: The content component is defined as a number of monetary units of a particular currency. It is a specific value, for example 12. One supplementary component

indicates the currency, expressed as a 3-letter alphabetic code value (i.e. USD=U.S. dollars, CAN=Canadian dollars). Another supplementary component identifies the version of the code list of the currency codes, i.e. the publication that states the USD stands for U.S. dollars, CAN stands for Canadian dollars, etc. Refer to the UN/CEFACT ebXML Core Components Technical Specification, version 1.9.

ASC X12  
- WORKING DRAFT -

## 5 Primitives

### 5.1 Primitive – Definition

A Primitive describes a single discrete piece of information. It is the lowest level of CICA. It is based on a single CCT. Each Primitive has one content component and zero or more supplementary components. Each content and supplementary component may have a number of restrictions placed with the primitive sub-class.

Multiple primitives may be derived from a single CCT.

A Primitive specifies a slightly more specific semantic purpose than that of the CCT, yet they remain highly reusable. A Primitive is created starting with a specific CCT and deriving a set of semantically specific usages for it. Primitives may be thought of as subclasses of core component types.

### 5.2 Primitive – Content

A Primitive consists of a single CCT content component and zero or more CCT supplementary components. To put this in object-oriented terms, each Primitive is a subclass of the CCT and inherits from the CCT its list of CCT Components. The Primitive is assigned a name different than the name of the CCT it derives from, reflecting its unique semantic intent.

### 5.3 Primitive – Use

A Primitive is used as a member of Components and Blocks (higher level constructs).

### 5.4 Primitive – Examples

The following are examples of Primitives:

- Weight Measure
- Total Amount
- Party Identifier
- Net Quantity

### 5.5 Primitive – Design Rules

#### 5.5.1 Semantic Rules:

- MUST be derived from a single Core Component Type (CCT).
- MAY be derived from the same Core Component Type as one or more other Primitives.
- SHOULD represent a single, discrete piece of information.
- SHOULD represent a unique piece of information across all Primitives.

## 5.5.2 Primitive Superclass Property Rules:

The primitive superclass MUST have the following characteristics:

### 5.5.2.1 Identifier

A primitive MUST have an identifier assigned by the X12 CICA Database. The identifier MUST be unique across database.

### 5.5.2.2 Primitive Superclass Name

A primitive MUST have a logical English language name. (Example: Name Text) The Primitive Name MUST be unique across all Primitives. The Primitive Name MUST comply with the X12.7 naming conventions.

### 5.5.2.3 Primitive Description

A Primitive SHOULD have a description of the meaning and purpose of the Primitive. Example: Free-form text that represent the name of a Party.

### 5.5.2.4 Primitive Core Component Type

A primitive MUST identify a single Core Component Type (CCT) that is based upon. Example: TextType

## 5.5.3 Primitive Subclass Property Rules:

The primitive subclass MUST have the following characteristics:

### 5.5.3.1 Primitive Subclass Name

A primitive MUST have a dictionary name for the primitive subclass. The primitive subclass name MUST be formed by appending a two-digit sequence number to the primitive superclass name.

### 5.5.3.2 Content Component Restriction

Each content component and supplementary component MUST be assigned a specific W3C XML Schema Language (or XSD) data type, and the allowable XSD restrictive facets supported by CICA restrictions allowed on the primitive content component as defined in section 4.1.1 of X12.7

### 5.5.3.3 Content Component Restriction Value

Each content component and supplementary component identified MUST have an appropriate value for the restrictive property

## 5.5.4 Primitive Usage Rules

- A primitive SHALL only be used as a member of a Block or a Component.
- When a primitive is used, it MUST be assigned a Usage Name.



## 6 Components

### 6.1 Component – Definition

A Component fully expresses either an identity or characteristic for a higher-level construct. They become a part of the definition of a Block, the next higher-level entity in the architecture. Components are a collection of two (2) or more Primitives. Components occupy the level above Primitives in the CICA architecture.

### 6.2 Component – Content

A Component is composed of two or more members, each of which is a Primitive. The term “member set” refers to the list of members that make up the Component. Each Component member is a Primitive and is uniquely defined by its usage name, not by the Primitive from which it is derived. In object-oriented terms, each Component member is “derived” from a Primitive.

### 6.3 Component – Use

A Component is used as a member of Blocks.

### 6.4 Component – Examples

The following examples are cases where a single identity or characteristic is conveyed by more than one piece of information:

- Person Name - consists of First Name, Middle Name, Last Name
- Mailing Address – consists of Street Address, City, State, etc.
- Some Components may express a range of values for some object. A range specifies, in some manner, a minimum value and a maximum value, and typically will be of the same data type (i.e. dates, amounts, text, etc.).

Examples:

- Date Range - consists of Start Date and End Date.
- Price Range – consists of a Minimum Price and a Maximum Price.
- Product Code Range – consists of a Start Product Code and an End Product Code (i.e. to restrict the list of products of interest by a range of code values).

Given the example:

C00001 <b>Component Name:</b> Person Name		
<b>Component XMLName:</b> PersonName		
<b>Component Description:</b> The name of an individual person.		
<b>Pos</b>	<b>PName</b>	<b>Usage Name</b>
01	NameText	First Name

02	NameText	Middle Name
03	NameText	Last Name

- This Component's member set consists of {First Name, Middle Name, Last Name}.
- Each of the following is a member of this Component: First Name, Middle Name, and Last Name.
- Each member inherits from the Primitive NameText.

## 6.5 Component – Design Rules

### 6.5.1 Component Semantic Rules

- MUST consist of at least two Primitives.
- MAY consist of more than two Primitives.
- MUST have a unique member set across all Components, as defined by the Usage Names.
- MAY have a member set that is a subset or superset of another Component's member set.
- MAY have more than one member that inherits from the same Primitive.
- MAY be used in one or more Blocks.
- MUST represent a specific identity or characteristic of an object.
- SHOULD represent a unique identity or characteristic across all Components.
- SHOULD have an abstract naming that is not specific to a single business context.

### 6.5.2 Component Member Semantic Rules

- MUST inherit from a single Primitive (identified by Primitive ID/Name)
- MAY inherit from the same Primitive as another member.
- MUST have a unique Usage Name within the Component.
- MAY have the same Usage Name as a member of a different Component.
- SHOULD represent a single, discrete piece of information that is a subset of the specific identity or characteristic represented by the Component as a whole.

## 6.5.3 Component Superclass Property Rules

### 6.5.3.1 Component Superclass Name

A component **MUST** have a logical English language name. (Example: Person Name) The component name **MUST** be unique across all Components. The component name **MUST** comply with the X12.7 naming conventions.

### 6.5.3.2 Component Description

A component **SHOULD** have a description of the meaning and purpose of the Component. Example: Full name of an individual person.

## 6.5.4 Component Member Properties

### 6.5.4.1 Component Member Usage Name

A component member **MUST** have a name that describes its usage within the component. (Example: First Name) The component usage **MUST** be unique within the Component.

### 6.5.4.2 Primitive Name

Name of the Primitive that this member inherits from. Example: Name  
Only filled in when used in Module [subclass properties]

## 6.5.5 Subclass Component Properties

### 6.5.5.1 Component Subclass Name

The dictionary name of the component subclass. The name **SHALL** be formed by appending a two-digit sequence number to the superclass component name.

### 6.5.5.2 Component Subclass XML Name

The name of the complex type that represents the component concrete subclass. The name is formed by appending a two-digit sequence number to the name of the parent abstract superclass component.

## 6.5.6 Subclass Component Member Properties

### 6.5.6.1 Primitive Subclass Name

Is as defined in section 4.1 on Primitive. **MUST** be the name of a primitive subclass that is derived from the abstract superclass primitive defined in the primitive abstract superclass member.

### 6.5.6.2 Requirements Flag

As defined in Section 2.2. Indicates the requirement restriction imposed on the member of the block member constraint, that is, the requirements on the presence of the member within the component-subclass.

**6.5.6.3 Minimum Occurs**

As defined in Section 2.2. Indicates the minimum number of occurrences permitted for the block constraint member, that is, the member within the component-subclass.

**6.5.6.4 Maximum Occurs**

As defined in Section 2.2. Indicates the maximum number of occurrences permitted for the component constraint member, that is, the member within the component-subclass.

**6.5.6.5 Content Restriction Flag**

As defined in Section 2.2. Indicates the content restriction applying to a subset of primitive members within the component-subclass.

**6.5.6.6 Primitive Subclass XML Name**

The name of the complex Type that represents the primitive concrete subclass. The name is formed by appending a two-digit sequence number to the name of the parent abstract superclass primitive.

## 7 Blocks

### 7.1 Block - Definition

Blocks specify a single Party, Location, Resource or Event, in a semantically precise, yet business context neutral manner. Blocks are semantically precise, because their composition is specific for the type of noun being specified. Blocks are business context neutral in that they specify a general noun, and are not specific to the specific noun. Blocks occupy the next level above Components in the CICA architecture. Each Block, like any noun, answers the questions "who", "what", "where", "when", or "why". The Block members specify either Identity or Characteristic information that apply to this entity.

Blocks are context-neutral, that is, they are independent of a specific business use, and are reusable in higher-level constructs.

Blocks play a pivotal role in CICA.

### 7.2 Block – Content

A Block is composed of one or more members, each of which specifies a particular Identity or Characteristic of the Block. Each Block Member is either the use of a Component or a Primitive. The Block Member is assigned a usage name.

Each Block Member is assigned a designator that indicates if it is required or optional. The minimum and maximum number of occurrences for each Block member is also specified. Detailed descriptions of these attributes are provided in the Attributes section.

### 7.3 Block – Use

A Block is used in two different places in CICA. First, sets of Blocks can be grouped together to create larger neutral constructs – Assemblies. These constructs are useful for creating related groupings of Blocks, such as when you need a Party with a Location. But, these are completely reusable and neutral, the same way Blocks are.

Blocks are also used to construct Modules, which are context specific implementable constructs.

As a standalone entity, a Block represents a logical grouping of related data items that can be referenced and reused as a single entity.

But, when a Block is placed into a Module, either directly or via an Assembly containing the Block, the Block takes on additional characteristics. The additional characteristics further define or restrict the allowable data content. For example, a Block instance can specify minimum and maximum lengths on individual data fields at the lowest level (i.e. Primitives), and it can specify which Block members are required and how many times they may (or must) occur. Some of the characteristics applied to the Block instance override those established to the abstract Block, others are applied only at the Block instance and are not part of the abstract Block definition.

**From a Technical Perspective**, a Block is an abstract super-class. Each Module use will require certain properties be assigned [min length, max length, max repeat, etc.]. Each different set of properties is a separate sub-class of the abstract super-class.

**From a Business Perspective**, when a Block is used in a Module, this is the point in time when its specific business use is known. For example, a Person Party Block can be used to create a Student module, in an education business context. Thus, a set of properties is assigned to Block, and all of the Block Members, as deemed necessary for the specific Student use. The next Block use, can either use the set of properties previously assigned, or create a new set of properties.

## 7.4 Block – Examples

The following examples are cases where a Block would be used:

- Person – consists of a Person Name, Social Security Number, Height, Weight, Hair Color, Eye Color, etc.
- Product – consists of a Product Identifier, Product Description, Product Class Code, etc.
- Location – consists of an address – Street Address, City, State, Postal Code, Country, etc.
- Account – consists of Account Number, Account Type, Account Status, etc.

Given the example:

**Block Name:** Person Party  
**Block XMLName:** Person Party  
**Block Type:** Party  
**Block Description:** Describes an individual person.

Pos	Prim/Comp Name	Usage Name	Use
01	Person Name	Person Name	I
02	IDNumber	Social Security Number	I
03	Description	Hair Color	C

- This Block's member set consists of {Person Name, Social Security Number, and Hair Color}.

- Each member inherits from the Component or Primitive represented by the ID value and Name indicated in the Member ID/Name column.

## 7.5 Block – Properties

The statements below define a Block. Note that the term “member set” refers to the list of members that make up the Block. Each member in the set is uniquely defined by its Usage Name (not by the Primitive or Component being used).

### 7.5.1 Block Semantic Rules

- MUST consist of at least one member.
- MAY consist of more than one member.
- MUST have a unique member set across all Blocks, as defined by the Usage Names.
- MAY have a member set that is a subset or superset of another Block’s member set.
- MAY have more than one member that is a usage of the same Primitive or Component.
- MAY be used in one or more Assemblies.
- MAY be used in one or more Modules.
- MUST represent a specific Party, Resource, Location, or Event.
- MUST be business context neutral and independent of specific business use.

### 7.5.2 Block Member Semantic Rules

- MUST be a use of a single Component or Primitive (identified by Member Name).
- MAY contain multiple usages of the same Component or Primitive, but each instance must have a different Usage Name.
- MUST have a unique Usage Name within the Block.
- MAY have the same Usage Name as a member in a different Block.
- MUST be designated as either an Identity or Characteristic (see definition below).
- MUST represent a single Identity or Characteristic of the entity the Block represents.

## 7.5.3 Block Property Rules, Abstract Superclass

### 7.5.3.1 BlockName

The logical/semantic name assigned to this Block. Example: Person Party

- MUST be unique across all Blocks.
- Must be compliant with the naming conventions specified in X12.7.

### 7.5.3.2 BlockXMLName

The physical/syntactic name assigned this Block. Example: PersonParty

- MUST be unique across all Blocks.

### 7.5.3.3 BlockType

Indicates the general type of this Block, for classification/reporting purposes. Valid values are:

**Party** — describes a person, organization, or other entity; i.e. answers a “who” question

**Resource** — describes a thing or object, i.e. answers a “what” question

**Location** — describes a place, i.e. answers a “where” question

**Event** — describes an event or activity, i.e. answers “when” and “what happened” questions, can be associated with a particular occurrence of some event, usually associated with a specific time or timeframe.

### 7.5.3.4 BlockDescription

Free-form text that describes the meaning and purpose of the Block. Example: Describes an individual person.

## 7.5.4 Each Block Member Property Rules, Abstract Superclass:

### 7.5.4.1 Position

Sequential number that indicates the position of the member within the Block. Example: 01 [Editor’s Note: Need to verify for consistency with X12.7 for all references of ‘position’]

### 7.5.4.2 Name

The name of the Component or Primitive that this member is derived from. Examples: Person Name, Name

### 7.5.4.3 UsageName

The name assigned to this member that reflects its usage within the Block. Example: Person Name.

### 7.5.4.4 XMLUsageName

The XML name that corresponds to the Usage Name above. Example: PersonName

**7.5.4.5 Use**

Indicates if this member is an Identity or Characteristic. Valid values are:

**I** = Identity

**C** = Characteristic

An Identity generally distinguishes one instance of this object from another, and is frequently unique across the universe of objects, although this is not a requirement. The entity may or may not follow a given numbering scheme or format. Examples of Identity include the following:

Block Type	Block Name	Identity Examples
Party	Person	Person Name, Social Security Num
Resource	Product	UPC Code
Location	Property	Parcel Number
Event	Graduation	Graduation Date

A Characteristic describes a particular quality or attribute of the object, much like an adjective describes a particular quality or attribute of a noun. Elements containing free-form text are more likely to be classified as Characteristic rather than Identity, although this is not a rule. Examples of Characteristic include the following:

Block Type	Block Name	Characteristic Examples
Party	Person	Eye Color
Resource	Product	Product Description
Location	Property	Property Description
Event	Graduation	Type of Degree (Attainment), i.e. Bachelor's.

**7.5.5 Block Subclass Properties**

- <block-subclass name> - The dictionary name of the block subclass. The name SHALL be formed by appending a two-digit sequence number to the superclass <block\_name>.
- <block-subclass XML name> - The name of the complexType that represents the block concrete subclass. The name is formed by appending a two-digit sequence number to the name of the parent abstract superclass block.

**7.5.6 Block Member Subclass Properties**

- <subclass name> - Name of the Primitive or Component Subclass. MUST be the name of a component subclass that is derived from the abstract superclass component defined in the block abstract superclass member.
- <requirements flag> - As defined in Section 2.2. Indicates the requirement restriction imposed on the member of the block member constraint, that is, the requirements on the presence of the member within the block-subclass.



- <min occurs> - As defined in Section 2.2. Indicates the minimum number of occurrences permitted for the block constraint member, that is, the member within the block-subclass.
- <max occurs> - As defined in Section 2.2. Indicates the maximum number of occurrences permitted for the block constraint member, that is, the member within the block-subclass.
- <content restriction flag> - As defined in Section 2.2. Indicates the content restriction applying to a subset of component or primitive members within the block-subclass.
- <usage XML name> - The name of the element representing the member primitive or component as used within the block.

ASC X12  
- WORKING DRAFT -

## 8 Assemblies

### 8.1 Assembly - Definition

Assemblies occupy the next level above Blocks in the CICA architecture. An Assembly represents a group of logically related party, resource, location, and/or event entities in a business document. Each Assembly can be thought of as a group of nouns that forms a reusable unit. Assemblies are context-neutral, that is, they are independent of a specific business use, and are reusable within other Assemblies as well as within the next higher-level construct – Modules.

There are three types of data relationships expressed by Assemblies:

- Consists-of: where the members operate together as a unit to create a larger reusable grouping.
- Kinds-of: where the members are peers in that they perform the same role.

### 8.2 Assembly – Content

An Assembly is composed of one or more members, each of which specifies a sub-group of the entire Assembly. Each Assembly member is associated with either another Assembly or a Block - to put this in object-oriented terms, each Assembly member is “inherited” or “derived” from an Assembly or a Block. The Assembly member is assigned a name, which can be the same as, or different than the name of the object it inherits from. Typically it is assigned a different name that reflects its specific usage; hence the term applied to the name of the member is “Usage Name”.

Each member is assigned a designator that indicates if it is required, optional, or “conditionally required”, i.e. depending on the existence of other members in the Assembly. The minimum and maximum number of occurrences for each member is also specified. Detailed descriptions of these attributes are provided in the Attributes section.

### 8.3 Assembly – Use

An Assembly is used as a member of other Assemblies and Modules, much the same way Blocks and Assemblies are used as members of Assemblies. As a standalone entity, an Assembly is *abstract*, i.e. it represents a logical grouping of related data items that can be referenced and reused as a single entity.

But, when an Assembly is placed into a Module, the Assembly takes on additional characteristics – in essence, a new *instance* of the *abstract* Assembly is created that fits the requirements for the particular Module it is being used in. The additional characteristics further define or restrict the allowable data content. For example, an Assembly instance can specify which members are required and how many times they may (or must) occur. The characteristics applied to the Assembly instance override those established to the abstract Assembly.

### 8.4 Assembly - Examples

The following entities would be represented by an Assembly:

- Delivery Party – consists of a Person and Address.
- Product Status – consists of a Product and Quantity Information.

- Patient Record – consists of a Person and Appointment Events
- Account History – consists of Account information and Transaction Events
- Contact Information – consists of contact information of various types – Phone, Email, Mailing, etc. (each is a different entity (noun)).

#### 8.4.1 Assembly Semantic Design Rules

Assemblies may only be composed of the following

- Two-or-more Blocks
- Two-or-more Assemblies
- One-or-more Blocks and one-or-more other Assemblies

Assemblies are made up of two or more blocks and/or assemblies.

Assemblies are Neutral constructs; therefore their contents and naming are abstract and industry independent.

There are two types of data relationships which exist which are expressed at the Assembly level,

Consists-of where the children operate together as a unit to create a larger reusable grouping. Examples include:

- A Person with a Location
- A Person with an Event
- A Location with an Event

Kinds-of where the children are peers in that they perform the same role, Examples include:

- A set of ways to specify how to contact, phone number, fax number, email, etc.
- A set of parties, one of which is performing a role, such as a person or an organization, which has acted in a role. Specifically, this applies in Healthcare where a Doctor or a Professional Corporation is collecting payment.

#### 8.4.2 An Assembly Member Semantic Rules

- MUST inherit from a single Assembly or Block (identified by ID).
- MAY inherit from the same Assembly or Block as another Member.
- MUST have a unique Usage Name within the Assembly.
- MAY have the same Usage Name as a member in a different Assembly.

- MUST be designated as either Required, Optional, Any, or eXclusive-or (see definition below).
- MUST be designated with a minimum and maximum number of occurrences. The maximum number of occurrences MUST be equal to or greater than the minimum number of occurrences.

ASC X12  
- WORKING DRAFT -

### 8.4.3 Assembly Superclass Properties

#### 8.4.3.1 AssemblyName

The logical/semantic name assigned to this Block.

- Must be a semantically descriptive name for the Assembly.
- Shall be used as the primary means of identification of assemblies in the DISA database.
- Shall be maintained as unique in the X12 XML standards development process.
- MUST comply with naming conventions in X12.7.

#### 8.4.3.2 AssemblyXMLName

The physical/syntactic name assigned this Assembly.

- MUST be unique.
- MUST comply with naming conventions in X12.7.

#### 8.4.3.3 AssemblyDescription

Free-form text that describes the meaning and purpose of the Assembly. Example: Describes the name and location of a person.

### 8.4.4 Assembly Superclass Member Properties

#### 8.4.4.1 Name

The name of the Assembly or Block that this member is derived from. Examples: Ship To, Person Party

#### 8.4.4.2 UsageName

The name assigned to this member that reflects its usage within the Assembly. Example: Ship To Party

#### 8.4.4.3 XMLUsageName

The XML name that corresponds to the Usage Name above. Example: ShipToParty

#### 8.4.4.4 Assembly Type

Each Assembly is formed to specify information about a primary type of information: Party, Resource, Event or Location. The Type of the first member within the Assembly is typically the same as the AssemblyType. [Editor's Note: Need to verify for consistency with X12.7]

P = Party

R = Resource

E = Event

L = Location

#### 8.4.5 Assembly Subclass Properties

- <assembly-subclass name> - The dictionary name of the assembly subclass. The name SHALL be formed by appending a two-digit sequence number to the superclass <assembly\_name>.
- <assembly-subclass XML name> - The name of the complexType that represents the assembly concrete subclass. The name is formed by appending a two-digit sequence number to the name of the parent abstract superclass assembly, <assembly XML name>.

#### 8.4.6 Assembly Subclass Member Properties

- For each Assembly member of the Assembly, <member assembly-subclass name> - The <assembly-subclass name> of a concrete subclass that is a member of the concrete subclass of the assembly. MUST be the name of a concrete subclass of the abstract superclass named by <member assembly name>.
- For each Block member of the Assembly, <block-subclass name> - MUST be the name of a block subclass that is derived from the abstract superclass block defined in the assembly abstract superclass member.
- <requirements flag> - As defined in Section 2.2.1 Indicates the requirement restriction imposed on the member of the assembly member constraint, that is, the requirements on the presence of the member within the assembly-subclass.
- <min occurs> - As defined in Section 2.2.2. Indicates the minimum number of occurrences permitted for the assembly constraint member, that is, the member within the assembly-subclass.
- <max occurs> - As defined in Section 2.2.3. Indicates the maximum number of occurrences permitted for the assembly member constraint, that is, the member within the assembly-subclass.
- <content restriction flag> - As defined in Section 2.2.4. Indicates the content restriction applying to a subset of members within the assembly-subclass.

#### 8.4.7 Assembly Design Guidelines

- There are two types of data relationships that exist, which are expressed at the Assembly level,

- *Consists-of*, where the children operate together as a unit to create a larger reusable grouping. Examples include:

A Person with a Location

A Person with an Event

A Location with an Event

- *Kinds-of*, where the children are peers in that they perform the same role, Examples include:

A set of ways to specify how to contact, phone number, fax number, email, etc.

A set of parties, one of which is performing a role, such as a person or an organization, which has acted in a role. Specifically, this applies in Healthcare where a Doctor or a Professional Corporation is collecting payment.

ASC X12  
- WORKING DRAFT -

## 9 Templates

### 9.1 Definition

A Template is a framework or skeleton of a business document for a particular business process requirement. The business process determines the Template's high-level composition and use. The Template defines the general structure and content of a business document, from which specific implementations of business documents are based. For example, an Event Based Invoice Template is created for a specific business process use of an Invoice. Its contents describe the general document used by parties to request payment for goods and/or services. Specific Invoice documents are developed based on that Invoice Template.

### 9.2 Content

A Template is composed of an ordered list of one or more members, called Slots. Slots are named using neutral terms for the logical business purpose they identify, and act as placeholders for Modules. A Template Slot is linked with a list of Modules, and each Module is selected to use in the Slot, under different business conditions. A Document is built by assigning a Module to each Template Slot.

A Template is divided into three sections:

- Header – applies to the entire business process and specifies the business context and parties to the business exchange.
- Detail – contains a set of data relevant to the business process of the message.
- Summary – summarizes the information contained in the detail (use of this section is generally discouraged).

### 9.3 Use

The Template provides a big picture view, of all the possible Modules used in each Slot. This construct is a fundamental innovation in the CICA document assembly model. All contextual uses of the Template can be viewed through this construct. The Template is conceptually the focal point of the architecture, bridging between “neutral” constructs below the line, and “implementable” constructs above the line, as shown in the figure below:

- WORKING DRAFT -



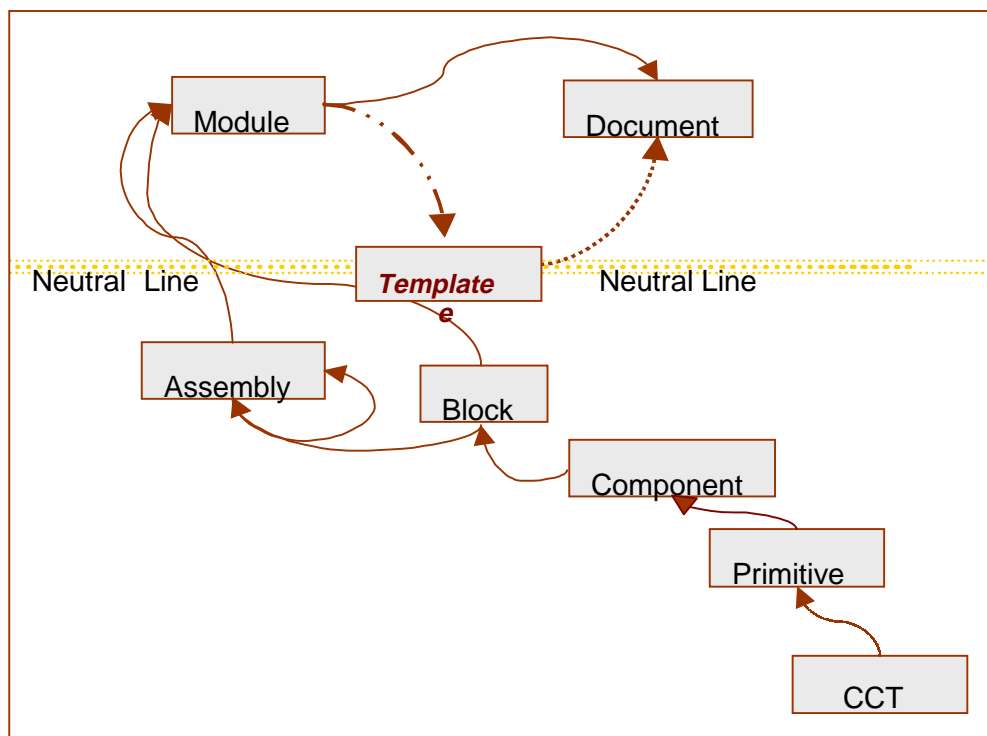


Figure 6. CICA Layers and Relationships

## 9.4 Examples

Templates can be created for the following general business documents:

- Event Invoice
- Bill of Lading
- Payment Request

If there is significant variation in the structure of a general business document, then multiple Templates could be created to accommodate these. For example, there are many types of Payment Requests, which can be categorized by the payment instrument to be used (ACH, wire transfer, credit card, etc.). A separate Template could be created for each, i.e.

- ACH Payment Request
- Wire Payment Request
- Credit Card Payment Request

## 9.5 Design Rules - Template

### 9.5.1 Template Semantic Design Rules

- A Template MAY be assigned to zero or more Documents.

- A Template MUST be specified for each Document.
- A Template MUST consist of one or more logically ordered and named Slots. These slots are reserved for Modules.
- New Templates can only be created when the business process specifies a new condition. This determination is made based on preconditions, trigger event, post conditions, and constraints.
- A Template MUST consist of one or more Slots, each of which is a member of either the Header, Detail, or Summary area.

### 9.5.2 Template Member Semantic Design Rules

- The Header area consists of Slots that contain information that applies to the entire business document and specifies the business context and parties to the business exchange.
- The Detail area consists of one or more Slots that contain information that describes the detail content of the message. The Slots in the Detail area may be repeated as a unit in a Document, i.e. the unit as a whole represents a single line item, and repeats for each line item in the message.
- The summary area consists of Slots that summarize information about the Detail area.
- The Slots pertaining to each AREA must be together.
- Areas must be ordered, Header first, Detail second, Summary last.
- Each area in the Template MAY be defined to have zero or more Slots.

### 9.5.3 Semantic Design Rules

- Each named Slot within the Template SHALL fulfill a specific function in the business process defined by the Template.
- Each named Slot SHALL, as its primary purpose, specify a single one of the following as related to the business process being identified:

**Who** (party: person / organization),

**What** (resource: financial, product [tangible or service])

**Where** (place: physical location)

**When** (event: past / present/ future).

- A Slot, as defined by its Slot Name, MAY appear in multiple Templates.
- Each Slot within a Template MUST accommodate one or more Modules.
- Slots SHALL be named using a neutral term for the logical business process they identify.

- Each Slot within a Template SHALL have a name that is unique.

## 9.5.4 Superclass Properties Design Rules

### 9.5.4.1 TemplateName

The logical/semantic name assigned to the Template.

- It MUST be unique across all entities within the CICA Architecture.

### 9.5.4.2 TemplateXMLName

The physical/syntactic name assigned to the Template; it corresponds to Name, above.

- It MUST be unique across all entities within the CICA Architecture.
- It MUST comply with the naming conventions in X12.7.
- It MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

### 9.5.4.3 TemplateDescription

Free-form text that describes the meaning and purpose of the Template.

### 9.5.4.4 TemplateFamily

Indicates the general category the Template falls within; it is a way to group “peer” Templates that cover the same general business purpose but are required because the associated documents vary in structure. For example, if separate Templates were needed for ACH Payment Request, Wires Payment Request, and Credit Card Payment Request, they would all be assigned the same Family of ‘Payment Request’.

### 9.5.4.5 BusinessProcess

Identifies of the Business Process to which this Template applies. The ebXML catalog of Common Business Processes [version 1.0] is a potential source. TAS will determine an initial set of values to choose amongst.

### 9.5.4.6 BusinessProcessFamily

Identifies of the family of Business Processes to which this Template applies. TAS will determine an initial set of values to choose amongst.

### 9.5.4.7 BusinessSubprocess

Identifies the sub process of the overall business process within which this Template is used. For example, Event-Based Invoicing is a subprocess of Invoicing. TAS will determine an initial set of values to choose amongst.

**9.5.4.8 BusinessSubprocessFamily**

Identifies a relationship amongst peer sub processes. For example if Event Based Invoicing were a Business Sub process, then it would be a member of the Invoicing family of subprocesses. TAS will determine an initial set of values to choose amongst.

**9.5.4.9 TriggeringEventDescription**

This describes the event in the business process being served that triggers the need to generate a message (Document) having this Template. This description may also include the range of expected responses to receipt of the generated message.

**9.5.4.10 ResponsibleSubCommittee**

Identifies the ASC-X12 Subcommittee with primary responsibility for maintenance of this Template.

**9.5.4.11 Context Category Value List**

As defined in Section 2.2.11. Indicates the set of contexts in which the template can be applied.

**9.5.5 Superclass Member Properties Design Rules****9.5.5.1 SlotName**

The name of the Slot within the Template. This term should be specific to the general business process, and not specific to an industry. Must conform with naming conventions

- It MUST be unique within the Template.

**9.5.5.2 SlotXMLName**

The physical/syntactic name assigned to the Slot; it corresponds to Name, above.

- It MUST be unique within the Template.
- It MAY be the same as a Slot of a different Template.
- It MUST comply with the naming conventions in X12.7.
- It MAY have a "default value" mechanically generated by the CICA Database Tool, which can be overridden.

**9.5.5.3 SlotPurpose**

Free-form text description describing the purpose of the Slot, from the perspective of the business process. It is a description of the purpose served by candidate Modules that fill the Slot.

**9.5.5.4 SlotDetailFlag**

Indicates which area in the Template the Slot belongs to.

H = Header

**D** = Detail

**S** = Summary

#### 9.5.5.5 Abstract requirements flag

Indicates whether or not the Slot is required (mandatory) or optional.

It **MUST** be one of the following values:

**M** = Required

**O** = Optional

#### 9.5.5.6 Abstract repeatability flag

As defined in Section 2.2. Indicates the repeatability restriction imposed on the slot of the template, that is, if the slot can be repeated within the template. This repeatability is independent of the detail repeating aspect.

#### 9.5.5.7 Context category value list

As defined in Section 2.11. Indicates the set of contexts in which the template can be applied.

#### 9.5.5.8 SlotModuleLinkage

The following identifiers must be supplied for each Module that can occupy the Slot.

**ModuleName** – the name of the Module as defined in the CICA Architecture. Must conform to the naming conventions in X12.7.

**ModuleUsageName.**

- **MUST** represent the logical/semantic name assigned to the Module that reflects the Module's usage within the Slot.

**SHOULD** be based on one of the following:

SlotName

ModuleName

Derived from the Module Name

**EXAMPLE:** If there were a general Template that could be used in many business processes, such as REGISTRATION, it could contain 3 Slots: Requestor, Registrar, and Registrant. The REGISTRATION Template could be used to register Births, Marriages, Students, or Toxic waste sites. When used in a specific industry, the Module names would be much more semantically descriptive than the Slot names, so the preference would be to derive the ModuleUsageName from the Module Name. Several similar Modules could exist, which are essentially identical except for detailed properties [minimum

length, requirement designation, etc.], resulting in Modules StudentA, StudentB and StudentC. Under these circumstances, the ModuleUsageName of Student would not equal the SlotName or the ModuleName, but rather would be derived from the ModuleName.

- MUST be unique across all Slots in the Template.
- MAY be the same as a Module in a different Template.

**ContextCategoryValue** – provides for a list of context rules that apply to the selection of this Module to occupy the Slot, as specified in section 2.2.11. These could potentially be used by a rules engine to programmatically determine which Module is appropriate for a Document that addresses a given business process. At this time, no such engine has been implemented; hence this entry is purely informational.

ASC X12  
- WORKING DRAFT -

## 10 Modules

### 10.1 Module - Definition

A Module represents a set of context-specific, related data that serves a specific purpose in a business document. Modules used within a specific business document satisfy the objective of the business document. Because Modules are above the neutral line, they always represent a detailed definition of the data, rather than an abstract definition.

### 10.2 Module – Content

A Module is composed of one or more members, each of which specifies a sub-group of the entire Module. Each Module member is derived from either an Assembly or a Block, i.e. the Assembly or Block definition that contains the additional characteristics that further define or restrict the allowable data content. This is because the Module is context-specific – above the neutral line – and therefore requires a fully qualified Assembly or Block rather than an abstract representation. To this point, each Module should be:

- Specific such that can be implemented without further qualification as to what each piece of information means
- Concise such that further explanation is not required in order to determine what pieces of information are used.

Each Module member is assigned a name that is different than the name of the object it is derived from. The name reflects its specific usage; hence the term applied to the name of the member is “Usage Name”.

Each Module member is assigned a designator that indicates if it is required, optional, and another designator conditional usage, i.e. depending on the existence of other members in the Module. The minimum and maximum number of occurrences for each member is also specified. Detailed descriptions of these attributes are provided in the Syntactic Design Rules.

### 10.3 Modules - Use

A Module occupies a Slot in a Template. A Module can be linked with multiple Templates and therefore can be used in multiple Documents, providing that the specific definition and characteristics of the Module apply in all cases.

### 10.4 Modules - Examples

The following are represented by a Module:

- Invoice Event – consists of invoice number, date, purchase order reference, and other details relevant to a particular invoice, i.e. conveys basic “header” information of the invoice.
- Buyer – consists of name, location, contact information, and other details relevant to the buying party.
- Invoice Line Goods – consists of a product, units, and other details relevant to a line item on an invoice, i.e. conveys the “detail” information of the invoice.

## 10.4.1 Module – Design Rules

The statements below define a Module. Note that the term “member set” refers to the list of members that make up the Module. Each member in the set is uniquely defined by its Usage Name (not by the Block or Assembly from which it is derived).

## 10.4.2 Semantic Design Rules

### 10.4.2.1 A Module

- MUST consist of at least one member.
- MAY consist of more than one member.
- MUST have a unique member set across all Modules, as defined by the Usage Names.
- MAY have a member set that is a subset or superset of another Module's member set.
- MAY have more than one member that inherits from the same Assembly or Block.
- MAY be designated to fill one or more Slots within a single Template or multiple Templates.
- MUST be a semantically unique entity of in terms of composition.
- MUST answer a specific semantic question within the business process (e.g., Who/What/When/Where/Why).
- MUST conform to the purpose of the Slot(s) in which it will be used.
- MAY be reusable within a business process and across business processes.
- MUST be specific such that further qualification is not required as to what each piece of information means.
- MUST be concise such that further explanation is not required in order to determine which pieces of information are used.

### - 10.4.2.2 A Module Member

- MUST be derived only from an Assembly or a Block.
- MAY be derived from the same Assembly or Block as another Member within the same Module.
- MUST have a unique Usage Name within the Module.
- MAY have the same Usage Name as a member in a different Module.
- MUST be designated as either Required or Optional.



- MUST be designated with a minimum and maximum number of occurrences.
- MAY have a maximum number of occurrences that is “unbounded”, meaning an unlimited number.
- MUST have a maximum number of occurrences that is equal to or greater than the minimum number of occurrences (unbounded is considered to be greater than or equal to any finite value).

### 10.4.3 Syntactic Design Rules

A **Module** is defined by the following:

#### 10.4.3.1 **ModuleName.**

- MUST be semantically descriptive.
- MUST uniquely identify the Module in the CICA database.
- MUST be unique across the entire CICA database (i.e. all levels).
- MUST conform to naming conventions defined in X12.7.

#### 10.4.3.2 **ModuleDescription.**

- MUST be a text paragraph describing the use, intent and overall use of the Module.

#### 10.4.3.3 **Module Type**

- MUST indicate which primary classification of information this Module conveys: Party, Resource, Event or Location. The Type of the first member within the Module is typically the same Type as the Module Type. [Editor's Note: Need to verify for consistency with X12.7 for all references of 'Module Class']

#### 10.4.3.4 **ResponsibleSubCommittee.**

- MUST designate the ASC X12 Subcommittee with primary responsibility for maintenance.

#### - 10.4.3.5 **ContextCategoryValue** -

Provides for a list of context rules that apply to the selection of this Module, as specified in section 2.2.11.

A **Module Member** is defined by the following:

#### 10.4.3.6 **UsageName.**

- MUST be unique within the Module. (redundant with Semantic rules)
- MUST be semantically descriptive of how the member is being used in the Module.

- MUST conform to the naming conventions in X12.7.

**10.4.3.7 AssemblyorBlock.**

- MUST be A or B to specify whether the member is an Assembly or a Block.

**10.4.3.8 RequirementsFlag.**

As defined in Section 2.2. Indicates the requirement restriction imposed on the member of the module member constraint, that is, the requirements on the presence of the member within the module concrete subclass.

**10.4.3.9 MinOccurs.**

As defined in Section 2.2. Indicates the minimum number of occurrences permitted for the module constraint member, that is, the member within the module concrete subclass.

**10.4.3.10 MaxOccurs.**

As defined in Section 2.2. Indicates the maximum number of occurrences permitted for the module constraint member, that is, the member within the module concrete subclass.

**10.4.3.11 Content restriction flag**

As defined in Section 2.2. Indicates the content restriction applying to a subset of block or assembly members within the module.

ASC X12  
- WORKING DRAFT -

# 11 Documents

## 11.1 Definition

A Document is a distinct, detailed message specification that is reflective of a context-specific business process. A Document is derived from a Template and its Slots, with context specific Modules occupying each Slot. A Document is defined by starting with a Template and for each Slot in the Template, selecting the single Module that is required for that particular Document.

## 11.2 Content

A Document is composed of the specific Modules that have been designated to fill each Slot in the Template that has been assigned to the Document.

## 11.3 Use

A Document definition is the complete specification of the message. From the Document, syntax specific representations are generated --- such as XML.

## 11.4 Examples

The following examples represent Documents:

- Delivery-Based Goods Invoice
- Statement-Based Service Invoice
- ACH Payment Request

## 11.5 Design Rules – Document

### 11.5.1 Semantic Design Rules

A **Document**:

- MUST represent an exchange of data that fulfills a single purpose in a business process.
- MUST contain a specific semantically complete definition.
- MUST be composed of:

A link to a specific Template.

A link to a specific Module for every Slot in the Template, made from the choices among the candidate Modules for each Slot in the Template.

A set of context references that drove the Module choices.

MUST conform to naming conventions defined in 4.10.

A **Document Member**:

- MUST be a specific Module must be designated to fill the Slot in the Template that is assigned to the Document.
- MUST fulfill a specific function in the business process defined by the Document.

A Module, as defined by its Module Name, may appear in multiple Documents.

## 11.5.2 Syntactic Design Rules

A **Document** is defined by the following:

### 11.5.2.1 DocumentName.

- MUST represent the logical/semantic name assigned to the Document.
- MUST be unique across all entities within the CICA Architecture.
- MUST conform to naming conventions defined in X12.7

### 11.5.2.2 DocumentXMLName.

- MUST represent the physical/syntactic name assigned to the Document; it corresponds to DocumentName, above.
- MUST be unique across all entities within the CICA Architecture.
- MUST comply with the naming conventions in X12.7.
- MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

### 11.5.2.3 DocumentDescription.

[Editor's Note: Need to verify for consistency with X12.7]

- MUST be free-form text that describes the meaning, use, and overall purpose of the Document.

### 11.5.2.4 DetailMaxOccurs.

[Editor's Note: Need to verify for consistency with X12.7]

- MUST indicate the maximum number of times the Detail Area can repeat.
- MUST be equal to the corresponding value in the specified Template, or a “hardening” of it (e.g., unbounded in the Template, and 25 here).

### 11.5.2.5 ResponsibleSubCommittee.

- MUST designate the ASC X12 Subcommittee with primary responsibility for maintenance

**11.5.2.6 ContextCategoryValue**

Provides for a list of context rules that apply to the selection of this Document [a Template with a selected collection of Modules], as specified in section 2.2.11.

**Document Member** is defined by the following:

**11.5.2.7 DocumentSlot.**

- MUST be the same as the SlotName in the specified Template.

**11.5.2.8 ModuleSlotXMLName.**

- MUST represent the physical/syntactic name assigned to the Slot within this Document.
- It MUST be unique across all Slots within the Document.
- It MAY be the same as a Slot in a different Document.
- It MUST comply with the naming conventions in X12.7.

**11.5.2.9 ModuleName.**

- MUST indicate the name of the Module designated to fill the Slot, as defined in the CICA Architecture.

**11.5.2.10 ModuleUsageName.**

- MUST represent the logical/semantic name assigned to the Module that reflects the Module's usage within the Slot.

SHOULD be based on one of the following:

SlotName

ModuleName

Derived from the Module Name

**EXAMPLE:** If there were a general Template that could be used in many business processes, such as REGISTRATION, it could contain 3 Slots: Requestor, Registrar, and Registrant. The REGISTRATION Template could be used to register Births, Marriages, Students, or Toxic waste sites. When used in a specific industry, the Module names would be much more semantically descriptive than the Slot names, so the preference would be to derive the ModuleUsageName from the Module Name. Several similar Modules could exist, which are essentially identical except for detailed properties [minimum length, requirement designation, etc.], resulting in Modules StudentA, StudentB and StudentC. Under these circumstances, the ModuleUsageName of Student would not equal the SlotName nor the ModuleName, but would be derived from the ModuleName.

- MUST be unique across all Slots in the Document.

- MAY be the same as a Module in a different Document.

#### 11.5.2.11 ContextCategory

#### 11.5.2.12 ContextCategoryValue

Provides for a list of context rules that apply to the selection of this Module within the Document, as specified in section 2.2.11.

#### 11.5.2.13 ReqDesignator.

- MUST indicate whether or not the Module is required (mandatory), or optional. This value overrides the corresponding value defined for the Slot in the Template.
- MUST be either R for Required, or O for Optional.
- MAY be the same as the defined for the Slot in the Template, or may be a “hardening” of it, i.e. Optional in the Template, and Required in the Document.
- MUST be compatible with the **MinOccurs** and **MaxOccurs** values for this member.

#### 11.5.2.14 MinOccurs.

- MUST indicate the minimum number of times this Module may repeat in this Slot of the Document. If the Module is in a Slot is in the Detail area, this represents the minimum number of times the Module may repeat within each occurrence/iteration of the entire Detail area. This value overrides the corresponding value defined for the Slot in the Template.
- MUST be equal to the corresponding value for the Slot in the associated Template, or a “hardening” of it - it must be equal to or greater than the value in the Template, i.e. 1 in the Template, and 2 in the Document.
- It MUST be a positive integer, i.e. equal to or greater than zero.
- It MUST be compatible with the **ReqDesignator**, i.e. if Optional, **MinOccurs** should be zero, if Required, **MinOccurs** should be one or greater.

#### 11.5.2.15 MaxOccurs.

- MUST indicate the maximum number of times this Module may repeat in this Slot of the Document. If the Modules is in a Slot is in the Detail area, this represents the maximum number of times the Module may repeat within each occurrence/iteration of the entire Detail area. This value overrides the corresponding value defined for the Slot in the Template.

- MUST be equal to the corresponding value for the Slot in the associated Template, or a “hardening” of it, i.e. unbounded in the Template, and 10 in the Document.
- MUST be either unbounded or a positive integer, and equal to or greater than the **MinOccurs** value.

ASC X12  
- WORKING DRAFT -

## APPENDIX A: CORE COMPONENTS CONTEXT CATEGORIES

In keeping with ASC X12's goal to align with the ebXML Core Components work, the following table and descriptive text are reproduced from Section 6.2.2 of the UN/CEFACT – ebXML Core Components Technical Specification, Part 1 (8 February 2002, Version 1.8). The UN/CEFACT – ebXML Core Components Technical Specification is copyrighted by UN/CEFACT and these excerpts are reproduced with that body's permission.

Copyright © UN/CEFACT 2002. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to UN/CEFACT except as required to translate it into languages other than English.

Note: The URL for the full document is

[http://www.unece.org/cefact/ebxml/ebXML\\_CCTS\\_Part1\\_V1-9.pdf](http://www.unece.org/cefact/ebxml/ebXML_CCTS_Part1_V1-9.pdf)

A comprehensive list of values must be specified for each context category. The ebXML CC specification has identified one or more available sources for each category. X12 plans to identify an "X12 selection" for the context categories that have multiple resources.

### Approved Context Categories

Table 1 contains the eight approved *Context Categories*.

- When describing a specific *Business Context*, a set of values will be assigned to the business situation being formally described.
- Applied *Business Context* will be from the list of approved context categories.

**Table 1. Approved Context Categories**

Business Process	The business process as described using the ebXML Catalogue of Common Business Processes as extended by the user.
Product Classification	Factors influencing semantics that are the result of the goods or services being exchanged, handled, or paid for, etc. (e.g. the buying of consulting services as opposed to materials)
Industry Classification	Semantic influences related to the industry or industries of the trading partners (e.g., product identification schemes used in different industries).
Geopolitical	Geographical factors that influence business semantics (e.g., the structure of an address).
Official Constraints	Legal and governmental influences on semantics (e.g. hazardous materials information required by law when shipping goods).
Business Process Role	The actors conducting a particular business process, as identified in the Catalogue of Common Business Processes.
Supporting Role	Semantic influences related to non-partner roles (e.g., data required by a third-party shipper in an order response going from seller to buyer.)



System Capabilities	This context category exists to capture the limitations of systems (e.g. an existing back office can only support an address in a certain form).
---------------------	--------------------------------------------------------------------------------------------------------------------------------------------------

## Business Process Context

In describing a business situation, generally the most important aspect of that situation is the business activity being conducted. *Business Process Context* provides a way to unambiguously identify the business activity. To ensure consistency with business process activities, it is important to use a common point of reference. The definitive point of reference for international standards is the UN/CEFACT *Catalogue of Common Business Processes*

- Assigned Business Process Contexts shall be from the standard hierarchical classification: provided as part of the UN/CEFACT Catalogue of Common Business Processes.
- Business Process Context values may be expressed as a single business process at any level, or may be expressed as a set of business processes at any level.
- Business Process Context values may be taken from extensions to the business processes described in the Catalogue of Common Business Processes as provided for in that document.
- When business process extensions are used, they shall include full information for each value sufficient to unambiguously identify which extension is providing the value used.

## Product Classification Context

The Product Classification Context describes those aspects of a business situation related to the goods or services being exchanged by, or otherwise manipulated, or concerned, in the business process. Recognised code lists exist that provide authoritative sources of product classification contexts.

- A single value or set of values may be used in a *Product Classification Context*.
- If a hierarchical system of values is used for *Product Classification Context*, then these values may be at any level of the hierarchy.
- If more than one classification system is being employed, an additional value specifying which classification scheme has supplied the values used shall be conveyed.
- Product classification context code values shall be taken from recognized code lists to include:
  - Universal Standard Product and Service Specification (UNSPSC)
    - Custodian: Electronic Commerce Code Management Association (ECCMA)
  - *Standard International Trade Classification* (SITC Rev .3)
    - Custodian: United Nations Statistics Division (UNSD)
  - *Harmonised Commodity Description and Coding System* (HS)
    - Custodian: World Trade Organization (WTO)
  - *Classification Of the purposes of non Profit Institutions serving households* (COPI)

- Custodian: UNSD (This provides a mapping between the first three.)

## Industry Classification Context

The Industry Classification Context provides a description of the industry or sub-industry in which the business process takes place.

- An *Industry Classification Context* may contain a single value or set of values at any appropriate level of the value hierarchy.
- The *Industry Classification Context* value hierarchy must be identified.
- *Industry Classification Context* code values shall be taken from recognised code lists to include:
  - *International Standard Industrial Classification* (ISIC) -- Custodian: UNSD
  - *Universal Standard Product and Service Specification* (UNSPSC) Top-level Segment [digits 1 and 2] used to define industry. --Custodian: ECCMA

### NOTE

There are many other industry classification schemes that may be used for Industry Classification Context.

## Geopolitical Context

Geopolitical Contexts allow description of those aspects of the business context that are related to region, nationality, or geographically based cultural factors.

- *Geopolitical Context* shall consist of appropriate continent, economic region, country, and region identifiers.
- *Geopolitical Regional Classification* may associate one or more values with any business message or component. are related to region, nationality, or geographically based cultural factors. country, and region identifiers. any business message or component.
- *Geopolitical Regional Classification* shall employ the following hierarchical structure:
  - Global
    - [Continent]
    - [Economic Region]
    - [Country] - ISO 3166.1
    - [Region] - ISO 3166.2
- At any level of the *Geopolitical Regional Classification* hierarchy, a value may be a single value, a named aggregate, or cross-border value.
- *Geopolitical Regional Classification* hierarchy values shall structured as follows:
  - **Single Value:** A single value indicating a single continent, economic region, country, or region, depending on position within the hierarchy.
  - **Named Aggregate:** A related group of values (which may themselves be single values, named aggregates, or cross-border pairs of values), which have been related and assigned a name. A named aggregate contains at least two values.

- **Cross-Border:** One or more pairs of values, designated *To*, *From*, or *Bi- directional*, indicating the direction of cross-border context. Values may be named aggregates or single values.

### EXAMPLE

The following example shows an extract of the basic, single-value hierarchy of recommended values, based on the common ISO 3166.1 *Country Codes*. (The value at the top of any hierarchy is always understood to be *Global*.)

Europe

Eastern Europe

AL – ALBANIA

AM – ARMENIA

- Points in the *Geopolitical Regional Classification* hierarchy shall be specified by the use of the node value, or by the full or partial path.
- The full path of the *Geopolitical Regional Classification* hierarchy must be used to understand the hierarchy when complex constructs are employed.
- A single-point specification is understood to inherit all of the properties of the single-value hierarchy except where otherwise specified.
- *Geopolitical Values* will be taken from ISO 3166.1 and 3166.2

## Official Constraints Context

The Official Constraints Context category describes those aspects of the business situation that result from legal or regulatory requirements and similar official categories. This category contains two distinct parts:

- Regulatory and Legislative. These are normally unilateral in nature and include such things as customs.
- Conventions and Treaties. These are normally bi- or multilateral agreements and as such are different from regulatory and legislative constraints.
- *The Official Constraints Context* will consist of at least two values:
  - Identification of the legal or other classification used to identify the context values.
  - Identification of the official constraint itself. These values may represent a hierarchical structure depending on the official constraints system being referenced.

Because there is no known global classification of all *Official Constraints Contexts* as used here, any implementation must provide a set of recognised official constraints classifications for use within the appropriate *Core Components* Registry implementation.

- Individual *Core Component* implementations shall register used official constraint classification schemes with the appropriate supporting *Core Components* Registry implementation.

## Business Process Role Context

The Business Process Role Context describes those aspects of a business situation that are specific to an actor or actors within the business process. Its values are taken from the set of Role values provided

by the Catalogue of Common Business Processes. A Business Process Role Context is specified by using a value or set of values from this source.

- *Business Process Role Context* values shall be taken from an approved list provided by the business process model library being employed.
- The UN/CEFACT *Catalogue of Common Business Processes* shall be the definitive source of *Business Process Role Context* values for all UN/CEFACT *Business Information Entities*.

## Supporting Role Context

The Supporting Role Context identifies those parties that are not active participants in the business process being conducted but who are interested in it. A Supporting Role Context is specified with a value or set of values from a standard classification.

- *Supporting Role Context* values shall be taken from the UN/EDIFACT *Code List for DE 3035 Party Roles*.

### NOTE

Users are cautioned that duplication exists in the current version of the required code list. UN/CEFACT will review this code list to clarify duplicates and identify non- *Supporting Role Context* values.

## System Capabilities Context

This category identifies a system, a class of systems or standard in the business situation. The System Capabilities Context requires a least one pair of values: an identification of the classification scheme being used and a value from that scheme. A valid System Capabilities Context may include more than one such pair of values.

- *Systems Capabilities Context* values shall consist of pairs of values. Each pair shall be comprised of an identification of the referenced classification scheme and the value(s) being employed.

### NOTE

There is no known classification of all types of information systems and standards. It is recommended that a mechanism for the registration of system and standard names be provided by the ebXML registry, as valid values for the *System Capabilities Context*.

**Table 2. Permissible Representation Terms**

Representation Term	Definition	Links to Core Component Type
<b>Amount</b>	A number of monetary units specified in a currency where the unit of currency is explicit or implied.	Amount. Type
<b>Code</b>	A character string (letters, figures or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of an attribute. Codes usually are maintained in code lists per attribute type (e.g. colour).	Code. Type
<b>Date</b>	A day within a particular calendar year (ISO 8601).	Date Time. Type
<b>Date Time</b>	A particular point in the progression of time (ISO 8601).	Date Time. Type
<b>Graphic</b>	A diagram, graph, mathematical curves, or similar representation	Graphic. Type
<b>Identifier</b>	A character string used to establish the identity of, and	Identifier. Type

	distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme.  [Note: Type shall not be used when a person or an object is identified by its name. In this case the <i>Representation Term</i> "Name" shall be used.]	
<b>Indicator</b>	A list of two, and only two, values that indicate a condition such as on/off; true/false etc. (synonym: "Boolean").	Indicator. Type
<b>Measure</b>	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE Rec. 20.	Measure. Type
<b>Name</b>	A word or phrase that constitutes the distinctive designation of a person, place, thing or concept.	Text. Type
<b>Percent</b>	A rate expressed in hundredths between two values that have the same unit of measure.	Numeric. Type
<b>Picture</b>	A visual representation of a person, object, or scene	Picture. Type
<b>Quantity</b>	A number of non-monetary units. It is associated with the indication of objects. Quantities need to be specified with a unit of quantity.	Quantity. Type
<b>Rate</b>	A quantity or amount measured with respect to another measured quantity or amount, or a fixed or appropriate charge, cost or value e.g. US Dollars per hour, US Dollars per Euro, kilometre per litre, etc.	Numeric. Type
<b>Text</b>	A character string generally in the form of words of a language.	Text. Type
<b>Time</b>	The time within a (not specified) day (ISO 8601).	Date Time. Type
<b>Value</b>	Numeric information that is assigned or is determined by calculation, counting or sequencing. It does not require a unit of quantity or a unit of measure	Numeric. Type

In addition to permissible representation terms for *Core Components*, there are also permissible representation terms for *Aggregate Core Components* and *Core Component Types*. Table 3 contains the permissible representation terms that apply to *Aggregate Core Components* or *Core Component Types*.

- The *Representation Term* for *Aggregate Core Components* or *Core Component Types* shall be one of the list of permissible *Aggregate Core components* or *Core Component Type Representation Terms*

**Table 3. Permissible Representation Terms for Aggregate Core Components or Core Component Types**

<b>Representation Term</b>	<b>Definition</b>	<b>Links to Core Component Type</b>
<b>Details</b>	The expression of the aggregation of <i>Core Components</i> to indicate higher leveled information entities	Not Applicable
<b>Type</b>	The expression of the aggregation of <i>Core Components</i> to indicate the aggregation of lower leveled information entities to become <i>Core Component Types</i> . All <i>Core Component Types</i> shall use this <i>Representation Term</i>	Not Applicable
<b>Content</b>	The actual content of an information entity. <i>Content</i> is the first information entity in a <i>Core Component</i>	Used with the <i>Content</i>

	<i>Type</i>	<i>Components of Core Component Types</i>
--	-------------	---------------------------------------------------

ASC X12  
- WORKING DRAFT -